O'REILLY

4th Edition
Covers Version 7.4

# Programming PHP

## Creating Dynamic Web Pages



ترجمة:
علي العمامي
fb.com/al3mamyali

Kevin Tatroe &
Peter MacIntyre
Foreword by Michael Stowe

# الفصل الأول مقدمة لـ PHP

PHP هي لغة بسيطة لكنها قوية مصممة لإنشاء محتوى HTML. يغطي هذا الفصل الخلفية الأساسية للغة PHP. يصف طبيعة وتاريخ PHP، والأنظمة الأساسية التي يعمل عليها، وكيفية تكوينها. ينتهي هذا الفصل بعرض PHP عمليًا، مع عرض سريع للعديد من برامج PHP التي توضح المهام الشائعة، مثل معالجة بيانات النموذج، والتفاعل مع قاعدة بيانات، وإنشاء رسومات.

**ماذا تفعل PHP؟**

يمكن استخدام PHP بطريقتين أساسيتين:

**البرمجة النصية من جانب الخادم**

تم تصميم PHP في الأصل لإنشاء محتوى ويب حيوي، ولا يزال ملائمًا لهذه المهمة. لتوليد HTML، تحتاج إلى محلل PHP وخادم ويب لإرسال ملفات المستندات المشفرة. أصبح PHP أيضًا شائعًا في إنشاء محتوى حيوي عبر اتصالات قاعدة البيانات ومستندات XML والرسومات وملفات PDF والمزيد.

**برمجة سطر الأوامر**

يمكن لـ PHP تشغيل نصوص برمجية من سطر الأوامر، تمامًا مثل Perl أو awk أو shell Unix. يمكنك استخدام البرامج النصية لسطر الأوامر لمهام إدارة النظام، مثل النسخ الاحتياطي وتحليل السجلات؛ حتى بعض البرامج النصية من نوع الوظيفة CRON يمكن القيام بها بهذه الطريقة (كمهام PHP غير مرئية).

في هذا الكتاب، نركز على العنصر الأول: استخدام PHP لتطوير محتوى ويب حيوي.

تعمل PHP على جميع أنظمة التشغيل الرئيسية، من متغيرات Unix (بما في ذلك Linux و FreeBSD و Ubuntu و Debian و Solaris) إلى Windows و macOS. يمكن استخدامه مع جميع خوادم الويب المعروفة، بما في ذلك خوادم Apache و Nginx و OpenBSD، على سبيل المثال لا الحصر؛ حتى البيئات السحابية مثل Azure و Amazon في ازدياد.

اللغة نفسها مرنة للغاية. على سبيل المثال، لا تقتصر على إخراج HTML أو ملفات نصية أخرى فقط - يمكن إنشاء أي تنسيق مستند. يحتوي PHP على دعم مدمج لتوليد ملفات PDF وصور GIF و JPEG و PNG.

واحدة من أهم ميزات PHP هو دعمها الواسع لقواعد البيانات. يدعم PHP جميع قواعد البيانات الرئيسية (بما في ذلك MySQL و PostgreSQL و Oracle و Sybase و MS-SQL و DB2 وقواعد البيانات المتوافقة مع ODBC)، وحتى العديد من قواعد البيانات الغامضة. حتى قواعد البيانات الأحدث مثل NoSQL مثل CouchDB و MongoDB مدعومة أيضًا. باستخدام PHP، يعد إنشاء صفحات الويب ذات المحتوى الحيوي من قاعدة البيانات أمرًا بسيطًا للغاية.

وأخيرًا، توفر PHP مكتبة من كود PHP لأداء المهام الشائعة، مثل تجريد قاعدة البيانات ومعالجة الأخطاء وما إلى ذلك، باستخدام ملحق PHP وملف التطبيق (PEAR). **PEAR**: هو إطار عمل ونظام توزيع لمكونات PHP قابلة لإعادة الاستخدام.

## نبذة تاريخية عن لغة PHP

تم تصميمها من قبل Rasmus Lerdorf لأول مرة من PHP في عام 1994، ولكن PHP الذي يستخدمه الناس اليوم يختلف تمامًا عن الإصدار الأولي. لفهم كيف وصلت PHP إلى حيث هي الآن، من المفيد معرفة التطور التاريخي للغة. إليكم هذه القصة بتعليقات وافرة ورسائل بريد إلكتروني من راسموس نفسه.

تطور PHP

هنا هو إعلان PHP 1.0 الذي تم نشره في مجموعة أخبار Usenet (comp.infosystems.www.authoring.cgi) في يونيو 1995:

الموضوع: إعلان: أدوات الصفحة الرئيسية الشخصية "Personal Home Page Tools" (أدوات PHP)

التاريخ: 08/06/1995

مجموعات أخبار: comp.infosystems.www.authoring.cgi

هذه الأدوات هي مجموعة من ثنائيات cgi الصغيرة الضيقة المكتوبة في C.

تؤدي عددا من الوظائف بما في ذلك:

- ❖ تسجيل الدخول يصل إلى صفحاتك في ملفات السجل الخاصة بك
- ❖ عرض معلومات السجل في الوقت الحقيقي
- ❖ توفير واجهة جميلة لمعلومات السجل هذه
- ❖ عرض معلومات الوصول الأخيرة مباشرة على صفحاتك

- ❖ عدادات وصول يومية وإجمالية كاملة
- ❖ حظر الوصول إلى المستخدمين على أساس المجال الخاص بهم
- ❖ حماية كلمة المرور للصفحات بناء على مجالات المستخدمين
- ❖ تتبع الوصول ** بناءً على عناوين البريد الإلكتروني للمستخدمين **
- ❖ تتبع عناوين URL المرجعية - دعم HTTP_REFERER
- ❖ يتضمن أداء جانب الخادم دون الحاجة إلى دعم الخادم لذلك
- ❖ القدرة على عدم تسجيل الوصول من مجالات معينة (أي الخاصة بك)
- ❖ إنشاء النماذج وعرضها بسهولة
- ❖ القدرة على استخدام معلومات النموذج في المستندات التالية

إليك ما لا تحتاجه لاستخدام هذه الأدوات:

- ❖ لا تحتاج إلى الوصول إلى الجذر - قم بالتثبيت في مجلد ~/public_html
- ❖ لا تحتاج إلى تمكين جانب الخادم في الخادم الخاص بك
- ❖ لا تحتاج إلى الوصول إلى Perl أو Tcl أو أي مترجم نص برمجي آخر
- ❖ لا تحتاج إلى الوصول إلى ملفات سجل httpd

الشرط الوحيد لهذه الأدوات للعمل هو أن لديك القدرة على تنفيذ برامج cgi الخاصة بك. اسأل مسؤول النظام إذا كنت غير متأكد مما يعنيه هذا.

تسمح لك الأدوات أيضًا بتنفيذ دفتر الزوار أو النموذج الذي يحتاج إلى كتابة المعلومات وعرضها للمستخدمين سابقا خلال دقيقتين تقريبًا.

الأدوات موجودة في المجال العام موزعة تحت رخصة جنو العمومية. نعم، هذا يعني أنها حرة!

للحصول على عرض توضيحي كامل لهذه الأدوات ، قم بتوجيه المتصفح

في: http://www.io.org/~rasmus

-

راسموس ليردورف

rasmus@io.org

http://www.io.org/~rasmus

... (تم تخطي هذا الجزء) وسيتم الرجوع لهذه الجزئية -بإذن الله- ...

# تثبيت PHP

كما ذكرنا، PHP متاح للعديد من أنظمة التشغيل والأنظمة الأساسية. لذلك، نشجعك على الرجوع إلى وثائق PHP للعثور على البيئة الأكثر ملاءمة للبيئة التي ستستخدمها واتباع تعليمات الإعداد المناسبة.

من وقت لآخر، قد ترغب أيضًا في تغيير طريقة تكوين PHP. للقيام بذلك، سيكون عليك تغيير ملف تكوين PHP وإعادة تشغيل خادم الويب (Apache). في كل مرة تقوم فيها بإجراء تغيير على بيئة PHP، سيتعين عليك إعادة تشغيل خادم الويب (Apache) حتى تدخل هذه التغييرات حيز التنفيذ.

عادةً ما يتم الاحتفاظ بإعدادات تكوين PHP في ملف يسمى php.ini. تتحكم الإعدادات الموجودة في هذا الملف في سلوك ميزات PHP، مثل معالجة الجلسة ومعالجة النماذج. تشير الفصول اللاحقة إلى بعض خيارات php.ini، لكن بشكل عام لا تتطلب الكود في هذا الكتاب تهيئة مخصصة. راجع وثائق PHP لمزيد من المعلومات حول تكوين php.ini.

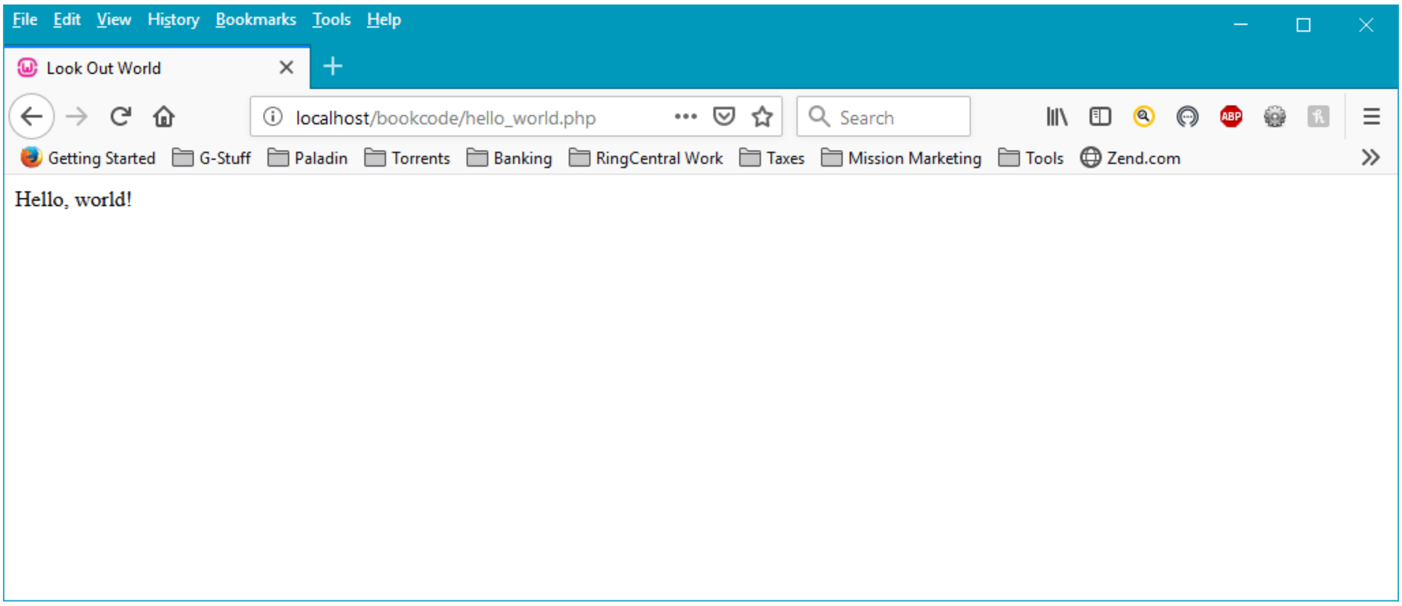
# جولة في PHP

صفحات PHP هي بشكل عام صفحات HTML تحتوي على أوامر PHP مضمنة فيها. هذا على عكس العديد من حلول صفحات الويب الديناميكية الأخرى، وهي عبارة عن برامج نصية تنشئ HTML. يعالج خادم الويب أوامر PHP ويرسل مخرجاتها (وأي HTML من الملف) إلى المتصفح. يوضح المثال 1-1 صفحة PHP كاملة.

مثال 1-1. hello_world.php

```
<html>
 <head>
 <title>Look Out World</title>
 </head>
 <body>
 <?php echo "Hello, world!"; ?>
 </body>
</html>
```

احفظ محتويات المثال 1-1 في ملف، hello_world.php، وقم بتوجيه المتصفح إليه. تظهر النتائج في الشكل 1-2.




الشكل 1-2. إخراج hello_world.php

ينتج أمر PHP echo إخراج (السلسلة) "Hello, world!" في هذه الحالة) يتم إدراجها في ملف HTML. في هذا المثال، يتم وضع كود PHP بين علامتي <?php و ?>. هناك طرق أخرى لوضع علامة على كود PHP الخاص بك - راجع الفصل 2 للحصول على وصف كامل.

# صفحة التكوين

دالة phpinfo() تنشئ صفحة HTML مليئة بالمعلومات حول كيفية تثبيت PHP وتهيئته حاليًا. يمكنك استخدامها لمعرفة ما إذا كان لديك امتدادات معينة مثبتة، أو ما إذا كان ملف php.ini قد تم تخصيصه. المثال 1-2 عبارة عن صفحة كاملة تعرض صفحة phpinfo().

المثال 1-2. باستخدام phpinfo()

```php
<?php phpinfo();?>
```

يوضح الشكل 1-3 الجزء الأول من ناتج المثال 1-2.

| PHP Version 7.4.0 | |
|---|---|
| System | Windows NT TOWERCASE 10.0 build 18362 (Windows 10) AMD64 |
| Build Date | Nov 27 2019 10:07:05 |
| Compiler | Visual C++ 2017 |
| Architecture | x64 |
| Configure Command | cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo" |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | enabled |
| Configuration File (php.ini) Path | C:\WINDOWS |
| Loaded Configuration File | D:\wamp64\bin\apache\apache2.4.41\bin\php.ini |
| Scan this dir for additional .ini files | (none) |
| Additional .ini files parsed | (none) |
| PHP API | 20190902 |
| PHP Extension | 20190902 |
| Zend Extension | 320190902 |
| Zend Extension Build | API320190902,TS,VC15 |
| PHP Extension Build | API20190902,TS,VC15 |
| Debug Build | no |
| Thread Safety | enabled |
| Thread API | Windows Threads |
| Zend Signal Handling | disabled |
| Zend Memory Manager | enabled |
| Zend Multibyte Support | provided by mbstring |
| IPv6 Support | enabled |
| DTrace Support | disabled |
| Registered PHP Streams | php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar |
| Registered Stream Socket Transports | tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3 |
| Registered Stream Filters | convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.* |

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.0, Copyright (c), by Zend Technologies
with Xdebug v2.8.0, Copyright (c) 2002-2019, by Derick Rethans

الشكل 1-3. الإخراج الجزئي لـ *phpinfo()*

## نماذج forms

يقوم المثال 1-3 بإنشاء نموذج ومعالجته. عندما يرسل المستخدم النموذج، يتم إرسال المعلومات المكتوبة في حقل الاسم مرة أخرى إلى هذه الصفحة عبر إجراء النموذج ['PHP_SELF']$_SERVER. يختبر كود PHP حقل name ويعرض ترحيباً إذا وجد واحدًا.

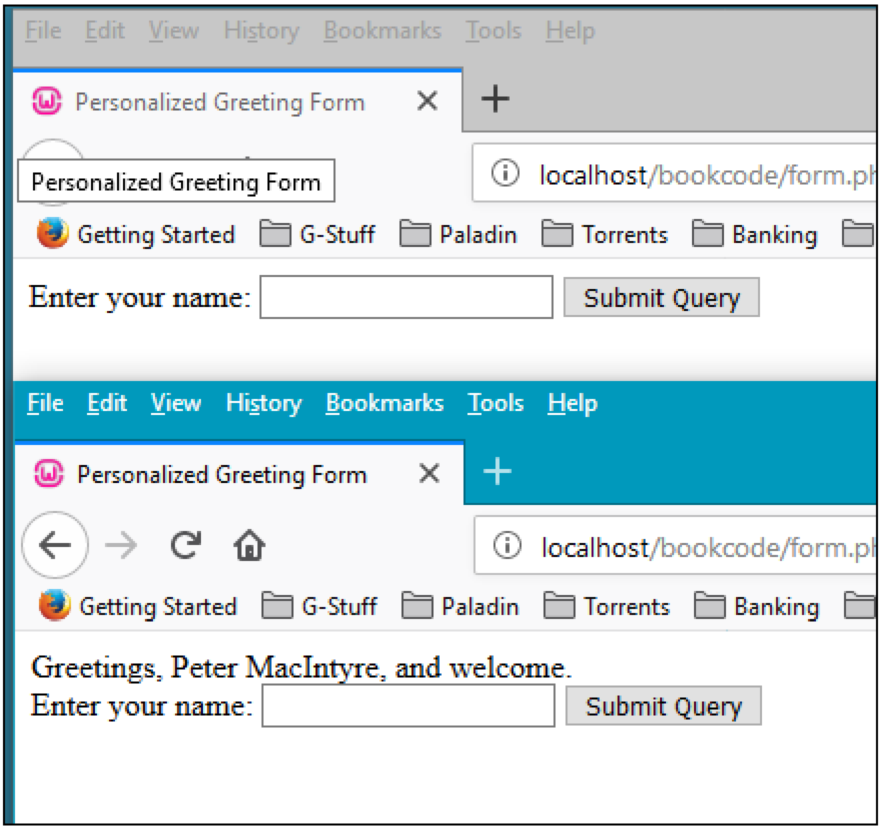
مثال 1-3. معالجة نموذج (form.php)

```
<html>
 <head>
 <title>Personalized Greeting Form</title>
 </head>

 <body>
 <?php if(!empty($_POST['name'])) {
 echo "Greetings, {$_POST['name']}, and welcome.";
 } ?>

 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="post">
 Enter your name: <input type="text" name="name" />
 <input type="submit" />
 </form>
 </body>
</html>
```

يظهر النموذج والرسالة في الشكل 1-4.




الشكل 1-4. صفحة النموذج والتحية

تصل برامج PHP إلى قيم النموذج بشكل أساسي من خلال متغيرات المصفوفة $_POST و $_GET.
الفصل 8 يناقش النماذج ومعالجة النماذج بمزيد من التفصيل.

# قواعد بيانات

يدعم PHP جميع أنظمة قواعد البيانات الشائعة، بما في ذلك قواعد البيانات المتوافقة مع MySQL و PostgreSQL و Oracle و Sybase و SQLite و ODBC. يوضح الشكل 1-5 جزءًا من استعلام قاعدة بيانات MySQL يتم تشغيله من خلال برنامج نصي PHP، ويعرض نتائج بحث عن كتاب على موقع لمراجعة الكتب. يسرد عنوان الكتاب وسنة نشر الكتاب ورقم ISBN للكتاب.

| These Books are currently available | | |
|---|---|---|
| Title | Year Published | ISBN |
| Executive Orders | 1996 | 0-425-15863-2 |
| Forward the Foundation | 1993 | 0-553-56507-9 |
| Foundation | 1951 | 0-553-80371-9 |
| Foundation and Empire | 1952 | 0-553-29337-0 |
| Foundation's Edge | 1982 | 0-553-29338-9 |
| I, Robot | 1950 | 0-553-29438-5 |
| Isaac Asimov: Gold | 1995 | 0-06-055652-8 |
| Rainbow Six | 1998 | 0-425-17034-9 |
| Roots | 1974 | 0-440-17464-3 |
| Second Foundation | 1953 | 0-553-29336-2 |
| Teeth of the Tiger | 2003 | 0-399-15079-X |
| The Best of Isaac Asimov | 1973 | 0-449-20829-X |
| The Hobbit | 1937 | 0-261-10221-4 |
| The Return of The King | 1955 | 0-261-10237-0 |
| The Sum of All Fears | 1991 | 0-425-13354-0 |
| The Two Towers | 1954 | 0-261-10236-2 |

الشكل 1-5. يتم تشغيل استعلام قائمة كتب MySQL من خلال نص PHP

يتصل الكود الموجود في المثال 1-5 بقاعدة البيانات، ويصدر استعلامًا لاسترداد جميع الكتب المتاحة (بجملة WHERE)، وينتج جدولًا كمخرج لجميع النتائج التي تم إرجاعها من خلال حلقة while.

**ملاحظة:**

كود SQL لقاعدة البيانات هذه موجود في library.sql الملف المتوفر. يمكنك إسقاط هذا الكود في MySQL بعد إنشاء قاعدة بيانات المكتبة والحصول على نموذج قاعدة البيانات تحت تصرفك لاختبار نموذج التعليمات البرمجية التالي بالإضافة إلى العينات ذات الصلة في الفصل 9.

مثال 1-4. الاستعلام عن قاعدة بيانات الكتب (booklist.php)

```
<?php

$db = new mysqli("localhost", "petermac", "password",
"library");

// make sure the above credentials are correct for your
environment
if ($db->connect_error) {
 die("Connect Error ({$db->connect_errno}) {$db-
>connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER
BY title";
$result = $db->query($sql);
```

```
?>
<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center"
border="1">
 <tr>
 <td colspan="4">
 <h3 align="center">These Books are currently
available</h3>
 </td>
 </tr>

 <tr>
 <td align="center">Title</td>
 <td align="center">Year Published</td>
 <td align="center">ISBN</td>
 </tr>
 <?php while ($row = $result->fetch_assoc()) { ?>
 <tr>
 <td><?php echo stripslashes($row['title']); ?></td>
 <td align="center"><?php echo $row['pub_year'];
?></td>
 <td><?php echo $row['ISBN']; ?></td>
 </tr>
 <?php } ?>
```

```
</table>


</body>
</html>
```

يحرك المحتوى الحيوي الذي توفره قاعدة البيانات الأخبار والمدونات ومواقع التجارة الإلكترونية في قلب الويب. مزيد من التفاصيل حول الوصول إلى قواعد البيانات من PHP موجودة في الفصل 9.

# الرسومات

باستخدام PHP، يمكنك بسهولة إنشاء الصور ومعالجتها باستخدام امتداد GD. يوفر المثال 1-5 حقل إدخال نص يتيح للمستخدم تحديد نص الزر. يأخذ ملف صورة زر فارغًا، ويقوم بتوسيط النص الذي تم تمريره على أنه "رسالة" معلمة GET. ثم يتم إرسال النتيجة إلى المتصفح كصورة PNG.

مثال 1-5. الأزرار الديناميكية (Graphic_example.php)

```
<?php
if (isset($_GET['message'])) {
 // load font and image, calculate width of text
 $font = dirname(__FILE__) . '/fonts/blazed.ttf';
 $size = 12;
 $image = imagecreatefrompng("button.png");
 $tsize      =      imagettfbbox($size,      0,      $font,
$_GET['message']);
```

```
 // center
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx) / 2;
 $y = (imagesy($image) - $dy) / 2 + $dy;

 // draw text
 $black = imagecolorallocate($im,0,0,0);
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$_GET['message']);

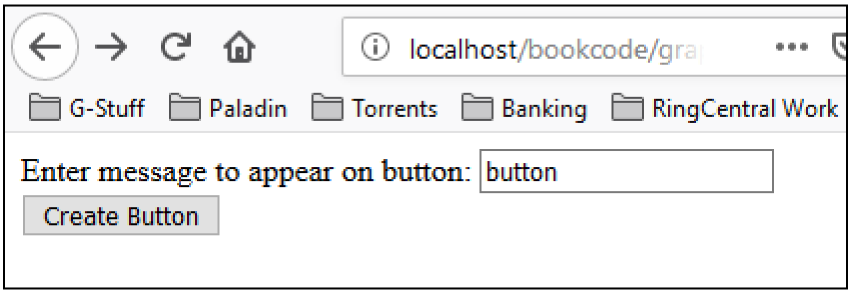
 // return image
 header("Content-type: image/png");
 imagepng($image);

 exit;
} ?>
<html>
 <head>
 <title>Button Form</title>
 </head>

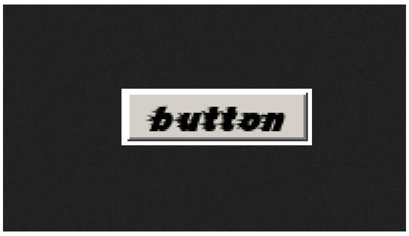
 <body>
 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Enter message to appear on button:
 <input type="text" name="message" /><br />
```

```
  <input type="submit" value="Create Button" />
  </form>
  </body>
</html>
```

يظهر النموذج الذي تم إنشاؤه بواسطة المثال 1-5 في الشكل 1-6. يظهر الزر الذي تم إنشاؤه في الشكل 1-7.




الشكل 1-6. شكل إنشاء زر




الشكل 1-7. تم إنشاء الزر

يمكنك استخدام GD لتغيير حجم الصور ديناميكيًا وإنتاج الرسوم البيانية وغير ذلك الكثير. تحتوي PHP أيضًا على عدة امتدادات لإنشاء مستندات بتنسيق PDF الشهير من Adobe. يغطي الفصل 10 إنشاء الصور الديناميكية بعمق، بينما يقدم الفصل 11 إرشادات حول كيفية إنشاء ملفات Adobe PDF.

# مالتالي

الآن بعد أن تعرفت على ما هو ممكن باستخدام PHP، فأنت جاهز لتعلم كيفية البرمجة باللغة. نبدأ بهيكلها الأساسي، مع التركيز بشكل خاص على الوظائف التي يحددها المستخدم، ومعالجة السلسلة، والبرمجة الموجهة للكائنات. ثم ننتقل إلى مناطق تطبيق محددة، مثل الويب وقواعد البيانات والرسومات و XML والأمان. ننتهي بمراجع سريعة للوظائف المدمجة والإضافات. أتقن هذه الفصول، وستكون قد أتقنت لغة PHP!

# الفصل الثاني: أساسيات اللغة

يقدم هذا الفصل جولة سريعة في لغة PHP الأساسية، تغطي الموضوعات الأساسية مثل أنواع البيانات والمتغيرات والعمليات وبيانات التحكم في التدفق. تتأثر لغة PHP بشدة بلغات البرمجة الأخرى، مثل Perl و C، لذلك إذا كانت لديك خبرة في استخدام هذه اللغات، فيجب أن تكون لغة PHP سهلة التعلم. إذا كانت PHP من أولى لغات البرمجة لديك، فلا داعي للذعر. نبدأ بالوحدات الأساسية لبرنامج PHP ونبني معرفتك من هناك.

## البنية المعجمية Lexical Structure

الهيكل المعجمي للغة البرمجة هو مجموعة القواعد الأساسية التي تحكم كيفية كتابة البرامج بتلك اللغة. إنه بناء الجملة الأقل مستوى للغة ويحدد أشياء مثل شكل أسماء المتغيرات، وما هي الأحرف المستخدمة للتعليقات، وكيف يتم فصل عبارات البرنامج عن بعضها البعض.

### الحساسية لحالة الأحرف

أسماء الفئات والدوال المعرفة من قبل المستخدم، وكذلك التركيبات والكلمات الرئيسية المضمنة (مثل echo، while، class، إلخ)، غير حساسة لحالة الأحرف. وبالتالي، فإن هذه الأسطر الثلاثة متساوية:

```
echo("hello, world");
ECHO("hello, world");
EcHo("hello, world");
```

من ناحية أخرى، فإن المتغيرات حساسة لحالة الأحرف. وهذا يعني أن $name و $NaME و $NAME هي ثلاثة متغيرات مختلفة.

## الجمل والفواصل المنقوطة

الجملة عبارة عن مجموعة من كود PHP يقوم بشيء ما. يمكن أن تكون بسيطة مثل تخصيص متغير أو معقدة complicated مثل: حلقة مع نقاط خروج متعددة. فيما يلي عينة صغيرة من جمل PHP، بما في ذلك استدعاءات الدوال، وبعض تخصيصات البيانات المتغيرة، وجمل if:

```
echo "Hello, world";
myFunction(42, "O'Reilly");
$a = 1;
$name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
```

تستخدم PHP الفاصلة المنقوطة للفصل بين الجمل البسيطة. لا تحتاج الجمل المركبة التي تستخدم الأقواس المتعرجة لتمييز كتلة من التعليمات البرمجية، مثل اختبار أو حلقة شرطية، إلى فاصلة منقوطة بعد قوس إغلاق. بخلاف اللغات الأخرى، في PHP، لا تعد الفاصلة المنقوطة قبل قوس الإغلاق اختيارية:

```
if ($needed) {
 echo "We must have it!"; // semicolon required here
} // no semicolon required here after the brace
```

ومع ذلك، فإن الفاصلة المنقوطة اختيارية قبل وسم إغلاق PHP:

```
<?php
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
echo "Hello, world" // no semicolon required before closing tag
?>
```

من الممارسات البرمجية الجيدة تضمين الفواصل المنقوطة الاختيارية، لأنها تسهل إضافة الكود لاحقًا.

## المسافة البيضاء وفواصل الأسطر

بشكل عام، لا تهم المسافات البيضاء في برنامج PHP. يمكنك نشر جمل عبر أي عدد من الأسطر، أو تجميع مجموعة من الجمل معًا في سطر واحد. على سبيل المثال، هذه الجملة:

```
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

يمكن أيضًا كتابتها بمزيد من المسافات:

```
raisePrices (
 $inventory ,
 $inflation ,
 $costOfLiving ,
```

```
 $greed
)  ;
```

أو بمسافة بيضاء أقل:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed
);
```

يمكنك الاستفادة من هذا التنسيق المرن لجعل كودك أكثر قابلية للقراءة (من خلال ترتيب المهام، والمسافة البادئة، وما إلى ذلك). يستفيد بعض المبرمجين البطيئين من هذا التنسيق الحر وينشئون كودًا غير قابل للقراءة تمامًا - وهذا غير مستحسن.

## تعليقات

تعطي التعليقات معلومات للأشخاص الذين قرأوا التعليمات البرمجية الخاصة بك، ولكن يتم تجاهلها بواسطة PHP في وقت التنفيذ. حتى إذا كنت تعتقد أنك الشخص الوحيد الذي سيقرأ كودك على الإطلاق، فمن الجيد تضمين التعليقات في شفرتك - في وقت لاحق، يمكن أن يبدو الكود الذي كتبته منذ شهور بسهولة كما لو أنه كتبه شخص غريب.

من الممارسات الجيدة أن تجعل تعليقاتك قليلة بما يكفي حتى لا تعترض طريق الكود نفسه ولكن وفيرة بما يكفي بحيث يمكنك استخدام التعليقات لمعرفة ما يحدث. لا تعلق على أشياء واضحة، لئلا تدفن التعليقات التي تصف أشياء صعبة. على سبيل المثال، هذا لا قيمة له:

```
$x = 17; // store 17 into the variable $x
```

بينما التعليقات على هذا التعبير العادي المعقد ستساعد كل من يحافظ على الكود الخاص بك:

```
// convert &#nnn; entities into characters


$text = preg_replace('/&#([0-9])+;/', "chr('\\1')",
$text);
```

توفر PHP عدة طرق لتضمين التعليقات في التعليمات البرمجية الخاصة بك، وكلها مستعارة من اللغات الحالية مثل C و ++C و Unix shell.

## تعليقات SHELL-STYLE

عندما تصادف PHP رمز علامة التجزئة (#) داخل الكود، فإن كل شيء بدءًا من علامة التجزئة إلى نهاية السطر أو نهاية قسم كود PHP (أيهما يأتي أولاً) يعتبر تعليقًا. توجد طريقة التعليق هذه في لغات البرمجة النصية لـ Unix shell وهي مفيدة في وضع تعليقات توضيحية على أسطر مفردة من التعليمات البرمجية أو عمل ملاحظات قصيرة.

نظرًا لأن علامة التجزئة مرئية على الصفحة، يتم استخدام تعليقات نمط الصدفة أحيانًا لتمييز كل التعليمات البرمجية:

```
#######################
## Cookie functions
#######################
```

يتم استخدامها في بعض الأحيان قبل سطر من التعليمات البرمجية لتحديد ما يفعله هذا الكود، وفي هذه الحالة يتم عادةً وضع مسافة بادئة لها على نفس مستوى الكود الذي تم إعداد التعليق له:

```
if ($doubleCheck) {
```

```
 # create an HTML form requesting that the user confirm the action
 echo confirmationForm();
}
```

غالبًا ما يتم وضع التعليقات القصيرة على سطر واحد من التعليمات البرمجية في نفس سطر الكود:

```
$value = $p * exp($r * $t); # calculate compounded interest
```

عندما تقوم بخلط كود HTML و PHP بإحكام، قد يكون من المفيد أن تقوم علامة PHP المغلقة بإنهاء التعليق:

```
<?php $d = 4; # Set $d to 4. ?> Then another <?php echo $d; ?>
Then another 4
```

## تعليقات ++C

عندما تصادف PHP شريحتين مائلتين (//) داخل الكود، فإن كل شيء بدءًا من الشرطات المائلة إلى نهاية السطر أو نهاية قسم الكود، أيهما يأتي أولاً، يعتبر تعليقًا. طريقة التعليق هذه مشتقة من ++C. والنتيجة هي نفس نمط تعليق الصدفة.

فيما يلي أمثلة للتعليقات على نمط الصدفة، أعيد كتابتها لاستخدام تعليقات ++C:

```
////////////////////////
// Cookie functions
////////////////////////

if ($doubleCheck) {
```

```
 // create an HTML form requesting that the user confirm
the action
 echo confirmationForm();
}
```

```
$value = $p * exp($r * $t); // calculate compounded
interest
```

```
<?php $d = 4; // Set $d to 4. ?> Then another <?php echo
$d; ?>
Then another 4
```

## تعليقات C

في حين أن التعليقات بنمط shell و ++C مفيدة للتعليق التوضيحي للكود أو لعمل ملاحظات قصيرة، تتطلب التعليقات الأطول أسلوبًا مختلفًا. لذلك، تدعم PHP تعليقات الكتلة التي يأتي تركيبها من لغة البرمجة C. عندما تصادف PHP شرطة مائلة متبوعة بعلامة النجمة (*/)، فإن كل شيء بعد ذلك، حتى يصادف علامة النجمة متبوعة بشرطة مائلة (/*) ، يعتبر تعليقًا. هذا النوع من التعليقات، على عكس تلك الموضحة سابقًا، يمكن أن يمتد إلى سطور متعددة.

في ما يلي مثال على تعليق متعدد الأسطر على نمط C:

```
/* In this section, we take a bunch of variables and
 assign numbers to them. There is no real reason to
 do this, we're just having fun.
*/
$a = 1;
```

```
$b = 2;
$c = 3;
$d = 4;
```

نظراً لأن التعليقات على النمط C لها علامات بداية ونهاية محددة، يمكنك دمجها بإحكام مع التعليمات البرمجية. يميل هذا إلى جعل قراءة التعليمات البرمجية أكثر صعوبة ولا يُنصح به:

```
/* These comments can be mixed with code too,
see? */ $e = 5; /* This works just fine. */
```

يمكن أن تستمر تعليقات نمط C، على عكس الأنواع الأخرى، بعد علامات PHP النهائية. فمثلا:

```
<?php
$l = 12;
$m = 13;
/* A comment begins here
?>
<p>Some stuff you want to be HTML.</p>
<?= $n = 14; ?>
*/
echo("l=$l m=$m n=$n\n");
?><p>Now <b>this</b> is regular HTML...</p>
l=12 m=13 n=
<p>Now <b>this</b> is regular HTML...</p>
```

يمكنك وضع مسافة بادئة للتعليقات كما تريد:

```
/* There are no
```

```
 special indenting or spacing
 rules that have to be followed, either.


 */
```

يمكن أن تكون التعليقات على النمط C مفيدة لتعطيل أقسام التعليمات البرمجية. في المثال التالي، قمنا بتعطيل الجملتين الثانية والثالثة، بالإضافة إلى التعليق المضمن، من خلال تضمينهما في تعليق الكتلة. لتمكين الكود، كل ما يتعين علينا القيام به هو إزالة علامات التعليق:

```
$f = 6;
/*
$g = 7; # This is a different style of comment
$h = 8;
*/
```

ومع ذلك، يجب أن تكون حريصًا على عدم محاولة دمج التعليقات:

```
$i = 9;
/*
$j = 10; /* This is a comment */
$k = 11;
Here is some comment text.
*/
```

في هذه الحالة، تحاول PHP (وتفشل) تنفيذ الجملة (non). إليك بعض نص التعليق وإرجاع خطأ.

## حرفية

الحرفي هو قيمة البيانات التي تظهر مباشرة في البرنامج. فيما يلي جميع القيم الحرفية في PHP:

```
2001
0xFE
1.4142
"Hello World"
'Hi'
true
null
```

## معرفات

المعرف هو مجرد اسم. في PHP، تُستخدم المعرفات لتسمية المتغيرات والدوال والثوابت والفئات. يجب أن يكون الحرف الأول للمعرّف حرف ASCII (أحرف كبيرة أو صغيرة)، أو حرف الشرطة السفلية (_)، أو أي من الأحرف بين ASCII 0x7F و ASCII 0xFF. بعد الحرف الأولي، هذه الأحرف والأرقام من 0 إلى 9 صالحة.

## أسماء المتغيرات

تبدأ أسماء المتغيرات دائمًا بعلامة الدولار ($) وتكون حساسة لحالة الأحرف. فيما يلي بعض أسماء المتغيرات الصالحة:

```
$bill
$head_count
$MaximumForce
```

```
$I_HEART_PHP
$_underscore
$_int
```

فيما يلي بعض أسماء المتغيرات غير القانونية:

```
$not valid
$|
$3wa
```

هذه المتغيرات كلها مختلفة بسبب حساسية حالة الأحرف:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

## أسماء الدوال

أسماء الدوال ليست حساسة لحالة الأحرف (تمت مناقشة الدوال بمزيد من التفصيل في الفصل الثالث). فيما يلي بعض أسماء الدوال الصالحة:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

تشير جميع أسماء الدوال هذه إلى نفس الدالة:

```
howdy HoWdY HOWDY HOWdy howdy
```

## أسماء الفصل CLASS

تتبع أسماء الفئات القواعد القياسية لمعرّفات PHP كما أنها ليست حساسة لحالة الأحرف. فيما يلي بعض أسماء الفئات الصالحة:

```
Person
account
```

اسم الفئة `stdClass` هو اسم فئة محجوز.

## الثوابت

الثابت هو معرف لقيمة لن تتغير، يمكن أن تكون القيم العددية (منطقية، وعدد صحيح، وعشري، وسلسلة) والمصفوفات ثوابت. بمجرد التعيين، لا يمكن أن تتغير قيمة الثابت. تتم الإشارة إلى الثوابت بواسطة معرفاتها ويتم تعيينها باستخدام دالة `define()`:

```
define('PUBLISHER', "O'Reilly Media");
echo PUBLISHER;
```

## الكلمات الرئيسية

الكلمة الرئيسية keyword (أو الكلمة المحجوزة reserved word) هي كلمة تخصصها اللغة لدوالها الأساسية - لا يمكنك إعطاء دالة أو فئة أو ثابت نفس اسم كلمة رئيسية. يسرد الجدول 2-1 الكلمات الرئيسية في PHP، والتي لا تتأثر بحالة الأحرف.

الجدول 2-1. الكلمات الرئيسية للغة PHP الأساسية

| | | |
|---|---|---|
| __CLASS__ | echo | insteadof |
| __DIR__ | else | interface |
| __FILE__ | elseif | isset() |
| __FUNCTION__ | empty() | list() |
| __LINE__ | enddeclare | namespace |
| __METHOD__ | endfor | new |
| __NAMESPACE__ | endforeach | or |
| __TRAIT__ | endif | print |
| __halt_compiler() | endswitch | private |
| abstract | endwhile | protected |
| and | eval() | public |
| array() | exit() | require |
| as | extends | require_once |
| break | final | return |
| callable | finally | static |
| case | for | switch |
| catch | foreach | throw |
| class | function | trait |
| clone | global | try |
| const | goto | unset() |
| continue | if | use |
| declare | implements | var |
| default | include | while |
| die() | include_once | xor |
| do | instanceof | yield |
| | | yield from |

بالإضافة إلى ذلك، لا يمكنك استخدام معرّف مماثل لدالة PHP المدمجة. للحصول على قائمة كاملة بها.

# أنواع البيانات

توفر PHP ثمانية أنواع من القيم أو أنواع البيانات. أربعة أنواع عددية (ذات قيمة واحدة): الأعداد الصحيحة، وأرقام عشرية، والسلاسل، والمنطقية. هناك نوعان من أنواع (التجميع) المركبة: المصفوفات والكائنات. النوعان المتبقيان هما نوعان خاصان: resource و NULL. تتم مناقشة الأرقام والمنطقية والموارد و NULL بالكامل هنا، بينما تعد السلاسل والمصفوفات والكائنات موضوعات كبيرة بما يكفي للحصول على الفصول الخاصة بها (الفصول 4 و 5 و 6 على التوالي).

## عدد صحيح

الأعداد الصحيحة هي أرقام صحيحة، مثل 1 و 12 و 256. يختلف نطاق القيم المقبولة وفقًا لتفاصيل النظام الأساسي الخاص بك ولكنه يمتد عادةً من 2،147،483،648- إلى 2،147،483،647+. على وجه التحديد، فإن النطاق يكافئ نطاق نوع البيانات الطويلة لمترجم سي الخاص بك. لسوء الحظ، لا يحدد معيار C النطاق الذي يجب أن يشتمل عليه هذا النوع الطويل، لذلك قد ترى في بعض الأنظمة نطاقًا صحيحًا مختلفًا.

يمكن كتابة القيم الحرفية الصحيحة بالنظام العشري أو الثماني أو الثنائي أو السادس عشري. يتم تمثيل القيم العشرية بتسلسل من الأرقام، بدون أصفار بادئة. قد يبدأ التسلسل بعلامة زائد (+) أو ناقص (-). إذا لم تكن هناك علامة، يفترض وجود إيجابي. تتضمن أمثلة الأعداد الصحيحة العشرية ما يلي:

```
1998
-641
+33
```

تتكون الأرقام الثمانية من صفر بادئ وسلسلة من الأرقام من 0 إلى 7. مثل الأرقام العشرية، يمكن أن تبدأ الأرقام الثمانية بعلامة زائد أو ناقص. فيما يلي بعض الأمثلة على القيم الثمانية والقيم العشرية المكافئة لها:

```
0755 // decimal 493
+010 // decimal 8
```

تبدأ القيم السداسية العشرية بـ 0x، متبوعة بسلسلة من الأرقام (0-9) أو بالحروف (A – F). يمكن أن تكون الأحرف كبيرة أو صغيرة ولكنها عادة ما تكون مكتوبة بأحرف كبيرة. كما هو الحال مع القيم العشرية والثمانية، يمكنك تضمين علامة بأرقام سداسية عشرية:

```
0xFF // decimal 255
0x10 // decimal 16
-0xDAD1 // decimal -56017
```

تبدأ الأعداد الثنائية بالرقم 0b، متبوعًا بتسلسل من الأرقام (0 و 1). كما هو الحال مع القيم الأخرى، يمكنك تضمين علامة في الأرقام الثنائية:

```
0b01100000 // decimal 96
0b00000010 // decimal 2
-0b10 // decimal -2
```

إذا حاولت تخزين متغير كبير جدًا بحيث لا يمكن تخزينه كعدد صحيح أو ليس عددًا صحيحًا، فسيتم تحويله تلقائيًا إلى رقم فاصلة عائمة.

استخدم الدالة ()is_int (أو الاسم المستعار ()is_integer الخاص بها) لاختبار ما إذا كانت القيمة عددًا صحيحًا:

```
if (is_int($x)) {
 // $x is an integer
}
```

## أرقام النقطة العائمة

تمثل أرقام الفاصلة العائمة (يشار إليها غالبًا بالأرقام "الحقيقية") قيمًا رقمية بأرقام عشرية. مثل الأعداد الصحيحة، تعتمد حدودها على تفاصيل جهازك. أرقام الفاصلة العائمة في PHP تكافئ نطاق نوع البيانات المزدوج لمترجم سي الخاص بك. عادة، يسمح هذا بالأرقام بين 1.7E−308 و 1.7E+308 بدقة 15 رقمًا. إذا كنت بحاجة إلى مزيد من الدقة أو نطاق أوسع من قيم الأعداد الصحيحة، فيمكنك استخدام امتدادات BC أو GMP.

تتعرف PHP على أرقام الفاصلة العائمة المكتوبة بتنسيقين مختلفين. هناك النوع الذي نستخدمه جميعًا كل يوم:

```
3.14
0.017
-7.1
```

لكن PHP تتعرف أيضًا على الأرقام في الترميز العلمي:

```
0.314E1 // 0.314*10^1, or 3.14
17.0E-3 // 17.0*10^(-3), or 0.017
```

قيم الفاصلة العائمة هي فقط تمثيلات تقريبية للأرقام. على سبيل المثال، في العديد من الأنظمة، يتم تمثيل 3.5 بالفعل كـ 3.4999999999. هذا يعني أنه يجب عليك الحرص على تجنب كتابة التعليمات البرمجية التي تفترض تمثيل أرقام الفاصلة العائمة بدقة كاملة، مثل المقارنة المباشرة بين قيمتين للفاصلة العائمة باستخدام ==. النهج العادي هو المقارنة بعدة خانات عشرية:

```
if (intval($a * 1000) == intval($b * 1000)) {
 // numbers equal to three decimal places
}
```

استخدم الدالة ()is_float (أو الاسم المستعار ()is_real الخاص بها) لاختبار ما إذا كانت القيمة رقم فاصلة عائمة:

```
if (is_float($x)) {
 // $x is a floating-point number
}
```

## سلاسل

نظراً لأن السلاسل شائعة جدًا في تطبيقات الويب، تتضمن PHP دعمًا على المستوى الأساسي لإنشاء السلاسل ومعالجتها. السلسلة عبارة عن سلسلة من الأحرف ذات طول عشوائي. يتم تحديد القيم الحرفية للسلسلة بعلامات اقتباس مفردة أو مزدوجة:

```
'big dog'
"fat hog"
```

يتم توسيع المتغيرات (interpolated) داخل علامات الاقتباس المزدوجة، بينما لا يتم توسيع المتغيرات داخل علامات الاقتباس المفردة:

```
$name = "Guido";
```

```
echo "Hi, $name <br/>";
echo 'Hi, $name';
Hi, Guido
Hi, $name
```

تدعم علامات الاقتباس المزدوجة أيضًا مجموعة متنوعة من عمليات الهروب من السلسلة، كما هو موضح في الجدول 2-2.

الجدول 2-2. الهروب من التسلسلات في سلاسل ذات علامات اقتباس مزدوجة

| Escape sequence | Character represented |
|---|---|
| \" | علامة تنصيص مزدوجة |
| \n | سطر جديد |
| \r | Carriage return |
| \t | مسافة جدول |
| \\ | Backslash |
| \$ | علامة الدولار |
| \{ | Left brace |
| \} | Right brace |
| \[ | Left bracket |
| \] | Right bracket |
| \0 through \777 | ASCII character represented by octal value |
| \x0 through \xFF | ASCII character represented by hex value |

نتعرف سلسلة ذات علامات اقتباس مفردة على \\ للحصول على شرطة مائلة عكسية و \' للحصول على اقتباس مفرد حرفي:

```
$dosPath = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O\'Reilly';
echo "$dosPath $publisher";
C:\WINDOWS\SYSTEM Tim O'Reilly
```

لاختبار ما إذا كانت سلسلتان متساويتين، استخدم عامل المقارنة == (مزدوج يساوي):

```
if ($a == $b) {
 echo "a and b are equal";
}
```

استخدم الدالة `is_string()` لاختبار ما إذا كانت القيمة سلسلة أم لا:

```
if (is_string($x)) {
 // $x is a string
}
```

توفر PHP عمليات حسابية ودوال لمقارنة السلاسل وتفكيكها وتجميعها والبحث عنها واستبدالها وتقليمها، بالإضافة إلى مجموعة من دوال السلاسل المتخصصة للعمل مع ترميز HTTP و HTML و SQL. نظراً لوجود العديد من دوال معالجة السلاسل، فقد خصصنا فصلًا كاملاً (الفصل 4) لتغطية جميع التفاصيل.

## قيمة منطقية

تمثل القيمة المنطقية قيمة *truth* - فهي توضح ما إذا كان الشيء صحيحًا أم لا. مثل معظم لغات البرمجة، تعرّف PHP بعض القيم على أنها true والبعض الآخر false. الصدق والخطأ يحددان نتيجة الكود الشرطي مثل:

```
if ($alive) { ... }
```

في PHP، يتم تقييم جميع القيم التالية على أنها false:

- الكلمة الرئيسية false
- العدد الصحيح 0
- قيمة النقطة العائمة 0.0
- السلسلة الفارغة ("") والسلسلة "0"
- مصفوفة بعناصر صفرية
- القيمة NULL

القيمة غير الصحيحة هي القيمة الصحيحة، بما في ذلك جميع قيم الموارد (التي سيتم وصفها لاحقًا في قسم "الموارد").

توفر PHP كلمات رئيسية true وfalse للتوضيح:

```
$x = 5; // $x has a true value
$x = true; // clearer way to write it
$y = ""; // $y has a false value
$y = false; // clearer way to write it
```

استخدم الدالة is_bool() لاختبار ما إذا كانت القيمة منطقية:

```
if (is_bool($x)) {
 // $x is a Boolean
}
```

## المصفوفات

تحتوي المصفوفة على مجموعة من القيم، والتي يمكنك تحديدها من خلال الموضع (رقم، مع كون الصفر هو الموضع الأول) أو بعض الأسماء التعريفية (سلسلة)، تسمى الفهرس الترابطي *"associative index"*:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";

$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

ينشئ بناء array() مصفوفة. إليك مثالين:

```
$person = array("Edison", "Wankel", "Crapper");
$creator = array('Light bulb' => "Edison",
 'Rotary Engine' => "Wankel",
 'Toilet' => "Crapper");
```

توجد عدة طرق للتكرار خلال المصفوفات، ولكن أكثرها شيوعًا هي حلقة foreach:

```
foreach ($person as $name) {
 echo "Hello, {$name}<br/>";
}

foreach ($creator as $invention => $inventor) {
 echo "{$inventor} invented the {$invention}<br/>";
}
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

يمكنك فرز عناصر المصفوفة بدوال الفرز المختلفة:

```
sort($person);
// $person is now array("Crapper", "Edison", "Wankel")

 asort($creator);
// $creator is now array('Toilet' => "Crapper",
// 'Light bulb' => "Edison",
// 'Rotary Engine' => "Wankel");
```

Use the is_array() function to test whether a value is an array:

```
if (is_array($x)) {
 // $x is an array
}
```

هناك دوال لإرجاع عدد العناصر في المصفوفة، وجلب كل قيمة في المصفوفة، وأكثر من ذلك بكثير. المصفوفات مغطاة بعمق في الفصل 5.

## Objects

يدعم PHP أيضًا البرمجة الشيئية (OOP). يعزز OOP التصميم المعياري النظيف، يبسط التصحيح والصيانة، ويساعد في إعادة استخدام الكود. الفئات هي اللبنات الأساسية للتصميم الموجه للكائنات. الفئة عبارة عن تعريف للبنية التي تحتوي على خصائص (متغيرات) وطرق (دوال). يتم تحديد الفئات باستخدام الكلمة الأساسية class:

```
class Person
{
 public $name = '';

 function name ($newname = NULL)
 {
 if (!is_null($newname)) {
 $this->name = $newname;
 }

 return $this->name;
 }
}
```

بمجرد تحديد فئة، يمكن إنشاء أي عدد من الكائنات منها باستخدام الكلمة الرئيسية new، ويمكن الوصول إلى خصائص الكائن وطرقه باستخدام ->:

```
$ed = new Person;
$ed->name('Edison');
echo "Hello, {$ed->name} <br/>";
$tc = new Person;
$tc->name('Crapper');
echo "Look out below {$tc->name} <br/>";
Hello, Edison
Look out below Crapper
```

استخدم الدالة `is_object()` لاختبار ما إذا كانت القيمة كائنًا:

```
if (is_object($x)) {
 // $x is an object
}
```

يصف الفصل 6 الفئات والكائنات بمزيد من التفصيل، بما في ذلك الوراثة inheritance والتغليف encapsulation والاستبطان introspection.

## مصادر أو موارد

توفر العديد من الوحدات العديد من الدوال للتعامل مع العالم الخارجي. على سبيل المثال، يحتوي كل ملحق قاعدة بيانات على الأقل على دالة للاتصال بقاعدة البيانات، ووظيفة للاستعلام عن قاعدة البيانات، ووظيفة لإغلاق الاتصال بقاعدة البيانات. نظرًا لإمكانية فتح اتصالات قاعدة بيانات متعددة في وقت

واحد، تمنحك دالة الاتصال شيئًا يمكن من خلاله تحديد هذا الاتصال الفريد عند استدعاء الاستعلام وإغلاق الدوال: مورد resource (أو مقبض *handle*).

لكل مورد نشط معرف فريد. كل معرف هو فهرس رقمي في جدول بحث PHP داخلي يحتوي على معلومات حول جميع الموارد النشطة. تحتفظ PHP بمعلومات حول كل مورد في هذا الجدول، بما في ذلك عدد المراجع (أو استخدامات) المورد في جميع أنحاء الكود. عندما يختفي المرجع الأخير لقيمة مورد، يتم استدعاء الملحق الذي أنشأ المورد لأداء مهام مثل تحرير أي ذاكرة أو إغلاق أي اتصال لهذا المورد:

```
$res = database_connect(); // fictitious database
connect function
database_query($res);

$res = "boo";
// database connection automatically closed because $res
is redefined
```

تظهر فائدة هذا التنظيف التلقائي بشكل أفضل في الدوال، عندما يتم تخصيص المورد لمتغير محلي. عندما تنتهي الدالة، تستعيد PHP قيمة المتغير:

```
function search() {
 $res = database_connect();
 database_query($res);
}
```

عند عدم وجود المزيد من الإشارات إلى المورد، يتم إيقافه تلقائيًا.

ومع ذلك، توفر معظم الإضافات إيقاف تشغيل محدد أو دالة إغلاق، ويُعتبر أسلوبًا جيدًا لاستدعاء هذه الدالة بشكل صريح عند الحاجة بدلاً من الاعتماد على النطاق المتغير لبدء عملية تنظيف الموارد.

استخدم الدالة `is_resource()` لاختبار ما إذا كانت القيمة موردًا:

```
if (is_resource($x)) {
 // $x is a resource
}
```

## عمليات الاسترجاعات Callbacks

عمليات الاسترجاعات هي دوال أو طرق كائن مستخدمة بواسطة بعض الدوال، مثل: `call_user_func()`. يمكن أيضًا إنشاء عمليات رد النداء بطريقة `create_function()` ومن خلال عمليات الإغلاق (الموضحة في الفصل 3):

```
$callback = function()
{
 echo "callback achieved";
};

call_user_func($callback);
```

## NULL

هناك قيمة واحدة فقط لنوع البيانات NULL. تتوفر هذه القيمة من خلال الكلمة الأساسية NULL غير حساسة لحالة الأحرف. تمثل القيمة NULL متغيرًا ليس له قيمة (مشابه لـ Perl's undef أو Python's None):

```
$aleph = "beta";
$aleph = null; // variable's value is gone
$aleph = Null; // same
$aleph = NULL; // same
```

استخدم الدالة is_null() لاختبار ما إذا كانت القيمة فارغة - على سبيل المثال، لمعرفة ما إذا كان للمتغير قيمة أم لا:

```
if (is_null($x)) {
 // $x is NULL
}
```

# المتغيرات

المتغيرات في PHP هي معرفات مسبوقة بعلامة الدولار ($). فمثلا:

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

قد يحمل المتغير قيمة من أي نوع. لا يوجد وقت ترجمة أو نوع وقت تشغيل التحقق من المتغيرات. يمكنك استبدال قيمة متغير بأخرى من نوع مختلف:

```
$what = "Fred";
$what = 35;
$what = array("Fred", 35, "Wilma");
```

لا توجد صيغة واضحة للتصريح عن المتغيرات في PHP. في المرة الأولى التي يتم فيها تعيين قيمة المتغير، يتم إنشاء المتغير في الذاكرة. بمعنى آخر، تعيين قيمة لمتغير يعمل أيضًا كإعلان. على سبيل المثال، هذا برنامج PHP كامل صالح:

```
$day = 60 * 60 * 24;
echo "There are {$day} seconds in a day.";
There are 86400 seconds in a day.
```

المتغير الذي لم يتم تعيين قيمته يتصرف مثل القيمة NULL:

```
if ($uninitializedVariable === NULL) {
 echo "Yes!";
}
Yes!
```

## المتغيرات المتغيرة Variable Variables

يمكنك الإشارة إلى قيمة المتغير الذي تم تخزين اسمه في متغير آخر عن طريق تقديم مرجع المتغير بعلامة دولار إضافية ($). فمثلا:

```
$foo = "bar";
$$foo = "baz";
```

بعد تنفيذ الجملة الثانية، المتغير $bar له القيمة "baz".

## مراجع متغيرة Variable References

في PHP، المراجع هي كيفية إنشاء أسماء مستعارة أو مؤشرات متغيرة. لجعل `$black` اسمًا مستعارًا للمتغير `$white`، استخدم:

```
$black =& $white;
```

فُقدت القيمة القديمة لـ `$black`، إن وجدت. بدلاً من ذلك، أصبح `$black` الآن اسمًا آخر للقيمة المخزنة في `$black`:

```
$bigLongVariableName = "PHP";
$short =& $bigLongVariableName;
$bigLongVariableName .= " rocks!";
```

```
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is PHP rocks!
Long is PHP rocks!
```

```
$short = "Programming $short";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

بعد التخصيص، يكون المتغيرين اسمين بديلين لنفس القيمة. لا يؤثر عدم ضبط متغير يحمل اسمًا مستعارًا على الأسماء الأخرى لقيمة هذا المتغير، ومع ذلك:

```
$white = "snow";
$black =& $white;
unset($white);
print $black;
snow
```

يمكن أن تُرجع الدوال قيمًا بالمرجع (على سبيل المثال، لتجنب نسخ سلاسل أو مصفوفات كبيرة، كما تمت مناقشته في الفصل 3):

```
function &retRef() // note the &
{
 $var = "PHP";
```

```
 return $var;
}

$v =& retRef(); // note the &
```

## نطاق المتغير

نطاق المتغير، الذي يتحكم فيه موقع إعلان المتغير، يحدد تلك الأجزاء من البرنامج التي يمكنها الوصول إليه. **هناك أربعة أنواع من نطاقات المتغير في PHP:** المحلي (local) والعالمي (global) والثابت (static) ومعلمات الدوال (function).

### النطاق المحلي

المتغير المعلن في دالة محلي لتلك الدالة. أي أنه مرئي فقط للكود في تلك الدالة (باستثناء تعريفات الدوال المتداخلة "function definitions")، لا يمكن الوصول إليه خارج الدالة. بالإضافة إلى ذلك، بشكل افتراضي، لا يمكن الوصول إلى المتغيرات المحددة خارج دالة (تسمى المتغيرات العامة) داخل الدالة. على سبيل المثال، إليك دالة تعمل على تحديث متغير محلي بدلاً من متغير عام:

```
function updateCounter()
{
 $counter++;
}

$counter = 10;
updateCounter();
```

```
echo $counter;
10
```

`$counter` داخل الدالة محلي لهذه الدالة لأننا لم نقل غير ذلك. تعمل الدالة على زيادة متغير `$counter` الخاص بها، والذي يتم إتلافه عند انتهاء الإجراء الفرعي. يظل `$counter` العالمي مضبوطًا على 10.

فقط الدوال يمكن أن توفر النطاق المحلي. على عكس اللغات الأخرى، في PHP لا يمكنك إنشاء متغير يكون نطاقه عبارة عن حلقة أو فرع شرطي أو أي نوع آخر من الكتل.

النطاق العالمي

المتغيرات المعلنة خارج دالة تكون عالمية. أي أنه يمكن الوصول إليها من أي جزء من البرنامج. ومع ذلك، فهي غير متوفرة بشكل افتراضي داخل الدالة. للسماح لدالة ما بالوصول إلى متغير عام، يمكنك استخدام الكلمة الأساسية `global` داخل الدالة لتعريف المتغير داخل الدالة. إليك كيفية إعادة كتابة دالة `updateCounter()` للسماح لها بالوصول إلى متغير `$counter` العام:

```
function updateCounter()
{
 global $counter;
 $counter++;
}

$counter = 10;
updateCounter();
```

```
echo $counter;
11
```

هناك طريقة أكثر تعقيدًا لتحديث المتغير العام وهي استخدام مصفوفة $GLOBALS الخاصة بـ PHP بدلاً من الوصول إلى المتغير مباشرةً:

```
function updateCounter()
{
 $GLOBALS['counter']++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

## المتغيرات الثابتة

المتغير الثابت يحتفظ بقيمته بين استدعاءات الدالة ولكنه مرئي فقط داخل تلك الدالة. تقوم بتعريف متغير ثابت باستخدام الكلمة الأساسية static. فمثلا:

```
function updateCounter()
{
 static $counter = 0;
 $counter++;

 echo "Static counter is now {$counter}<br/>";
```

```
}

$counter = 10;
updateCounter();
updateCounter();

echo "Global counter is {$counter}";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

## معلمات الدوال

كما سنناقش بمزيد من التفصيل في الفصل 3، يمكن أن يكون لتعريف الدالة معلمات مسمى:

```
function greet($name)
{
 echo "Hello, {$name}";
}

greet("Janet");
Hello, Janet
```

معلمات الدالة محلية، مما يعني أنها متاحة فقط داخل دوالها. في هذه الحالة، لا يمكن الوصول إلى `$name` من خارج `greet()`.

## جمع القمامة Garbage Collection

تستخدم PHP حساب المراجع *"reference counting"* والنسخ عند الكتابة *"copy-on-write"* لإدارة الذاكرة. يضمن النسخ عند الكتابة عدم إهدار الذاكرة عند نسخ القيم بين المتغيرات، ويضمن حساب المرجع إعادة الذاكرة إلى نظام التشغيل عندما لا تكون هناك حاجة إليها.

لفهم إدارة الذاكرة في PHP، يجب أن تفهم أولاً فكرة جدول الرموز *"symbol table"*. يتكون المتغير من جزأين - اسمه (على سبيل المثال، `$name`) وقيمته (مثل: `"Fred"`). جدول الرموز هو مصفوفة تقوم بتعيين أسماء المتغيرات لمواضع قيمها في الذاكرة.

عندما تنسخ قيمة من متغير إلى آخر، لا تحصل PHP على ذاكرة أكبر لنسخة من القيمة. بدلاً من ذلك، تقوم بتحديث جدول الرموز للإشارة إلى أن "كلا المتغيرين هما اسمان لنفس الجزء من الذاكرة". لذا فإن الكود التالي لا يُنشئ في الواقع مصفوفة جديدة:

```
$worker = array("Fred", 35, "Wilma");
$other = $worker; // array isn't duplicated in memory
```

إذا قمت بتعديل أي من النسختين لاحقًا، فإن PHP تخصص الذاكرة المطلوبة وتقوم بالنسخة:

```
$worker[1] = 36; // array is copied in memory, value
changed
```

من خلال تأخير التخصيص والنسخ، توفر PHP الوقت والذاكرة في الكثير من المواقف. هذه نسخة عند الكتابة.

كل قيمة يشير إليها جدول الرموز لها عدد مرجعي *"reference count"*، وهو رقم يمثل عدد الطرق المتاحة للوصول إلى تلك القطعة من الذاكرة. بعد التعيين الأولي للمصفوفة إلى `$worker` و `$worker` إلى `$other`، فإن المصفوفة المشار إليها بإدخالات جدول الرموز لـ `$worker` و `$other` تحتوي على عدد مرجعي يبلغ 2[1]. بمعنى آخر، يمكن الوصول إلى هذه الذاكرة بطريقتين: من خلال `$worker` أو `$other`. ولكن بعد تغيير `$worker[1]`، تُنشئ PHP مصفوفة جديدة لـ `$worker`، ويكون عدد المراجع لكل مصفوفة 1 فقط.

عندما يخرج متغير عن النطاق في نهاية دالة، مثل معلمات الوظيفة والمتغيرات المحلية، فإن العدد المرجعي لقيمته ينخفض بمقدار واحد. عندما يتم تعيين قيمة لمتغير في منطقة مختلفة من الذاكرة، يتم تقليل العدد المرجعي للقيمة القديمة بمقدار واحد. عندما يصل العد المرجعي لقيمة إلى 0، يتم تحرير ذاكرتها. هذا هو العد المرجعي.

العد المرجعي هو الطريقة المفضلة لإدارة الذاكرة. احتفظ بالمتغيرات محلية للوظائف، وقم بتمرير القيم التي تحتاج الوظائف للعمل عليها ، ودع حساب المرجع يتولى إدارة الذاكرة. إذا كنت تصر على محاولة الحصول على مزيد من المعلومات أو التحكم في تحرير قيمة المتغير، فاستخدم الدالة `isset()` و `unset()`.

لمعرفة ما إذا تم تعيين متغير على شيء ما - حتى السلسلة الفارغة - استخدم `isset()`:

```
$s1 = isset($name); // $s1 is false
$name = "Fred";
$s2 = isset($name); // $s2 is true
```

**استخدم** `unset()` **لإزالة قيمة المتغير:**

```
$name = "Fred";
unset($name); // $name is NULL
```

# التعبيرات والعمليات الرياضية

التعبير "*expression*" هو جزء من كود PHP يمكن تقييمه لإنتاج قيمة. أبسط التعبيرات هي القيم والمتغيرات الحرفية. يتم تقييم القيمة الحرفية لنفسها، بينما يقيم المتغير القيمة المخزنة في المتغير. يمكن تكوين تعبيرات أكثر تعقيدًا باستخدام تعبيرات وعوامل بسيطة.

تأخذ العملية "*operator*" بعض القيم (المعاملات) ويفعل شيئًا (على سبيل المثال، يجمعهم معًا). تتم كتابة العوامل أحيانًا كرموز ترقيم -على سبيل المثال، + و - مألوف لنا من الرياضيات. تقوم بعض العمليات الرياضية بتعديل معاملاتهم، بينما لا يقوم معظمهم بذلك.

يلخص الجدول 2-3 عوامل التشغيل في PHP، والتي تم استعارة العديد منها من C و Perl. يعطي العمود المسمى "P" أسبقية المشغل "operator's precedence"، يتم سرد العوامل بترتيب الأسبقية، من الأعلى إلى الأدنى. يعطي العمود المسمى "A" ارتباط المشغل "operator's associativity"، والذي يمكن أن يكون L (من اليسار إلى اليمين)، أو R (من اليمين إلى اليسار)، أو N (غير ارتباط "nonassociative").

الجدول 2-3. عوامل تشغيل PHP

| P | A | المشغل أو العامل | العملية |
|---|---|---|---|
| 24 | N | `clone, new` | إنشاء كائن جديد |
| 23 | L | [ | Array subscript |

| P | A | المشغل أو العامل | العملية |
| --- | --- | --- | --- |
| 22 | R | ** | الأس |
| 21 | R | ~ | Bitwise Not |
| | R | ++ | زيادة |
| | R | -- | نقصان |
| | R | (int), (bool), (float), (string ), (array), (object), (unset) | Cast |
| | R | @ | Inhibit errors |
| 20 | N | instanceof | Type testing |
| 19 | R | ! | ليس (المنطقي) |
| 18 | L | * | الضرب |
| | L | / | القسمة |
| | L | % | باقي القسمة |
| 17 | L | + | الجمع |

| P | A | المشغل أو العامل | العملية |
|---|---|---|---|
| | L | – | الناقص |
| | L | . | تسلسل السلسلة |
| 16 | L | << | Bitwise shift left |
| | L | >> | Bitwise shift right |
| 15 | N | <, <= | أقل من، أقل من أو يساوي |
| | N | >, >= | أكبر من، أكبر من أو يساوي |
| 14 | N | == | المساواة في القيمة |
| | N | !=, <> | عدم المساواة |
| | N | === | المساواة في النوع والقيمة |
| | N | !== | لايساوي النوع والقيمة |
| | N | <=> | تُرجع عددًا صحيحًا بناءً على مقارنة معاملين: |

| P | A | المشغل أو العامل | العملية |
|---|---|---|---|
| | | | 0 عندما يكون اليسار واليمين متساويين، و -1 عندما يكون اليسار أقل من اليمين، و 1 عندما يكون اليسار أكبر من اليمين. |
| 13 | L | & | Bitwise AND |
| 12 | L | ^ | Bitwise XOR |
| 11 | L | \| | Bitwise OR |
| 10 | L | && | و (المنطقي) |
| 9 | L | \|\| | أو (المنطقي) |
| 8 | R | ?? | مقارنة |
| 7 | L | ?: | عامل شرطي |
| 6 | R | = | Assignment |
| | R | +=, −=, *=, /=, .=, %=, &=, \|=, ^=, ~=, <<=, >>= | Assignment with operation |

| P | A | المشغل أو العامل | العملية |
|---|---|---|---|
| 5 | | `yield from` | العائد من |
| 4 | | `yield` | العائد |
| 3 | L | and | و (المنطقي) |
| 2 | L | xor | Logical XOR |
| 1 | L | or | أو (المنطقي) |

## عدد المعاملات

معظم العوامل في PHP هي عوامل ثنائية؛ يقومون بدمج معاملين (أو تعبيرين) في تعبير واحد أكثر تعقيدًا. تدعم PHP أيضًا عددًا من العوامل الأحادية، والتي تحول تعبيرًا واحدًا إلى تعبير أكثر تعقيدًا. أخيرًا، تدعم PHP عددًا قليلاً من العوامل الثلاثية التي تدمج العديد من التعبيرات في تعبير واحد.

## أسبقية المعامل

يعتمد الترتيب الذي يتم من خلاله تقييم العوامل في التعبير على أسبقيتها النسبية. على سبيل المثال، قد تكتب:

2 + 4 * 3

كما ترى في الجدول 2-3، فإن عوامل الضرب والجمع لها أسبقية مختلفة، الضرب أعلى من الجمع. إذن، يتم الضرب قبل عملية الجمع، ويكون الناتج 2 + 12، أو 14 كإجابة. إذا تم عكس أسبقية الجمع والضرب، فستكون الإجابة 6 * 3 أو 18.

لفرض ترتيب معين، يمكنك تجميع المعاملات باستخدام عامل التشغيل المناسب بين قوسين. في مثالنا السابق، للحصول على القيمة 18، يمكنك استخدام هذا التعبير:

```
(2 + 4) * 3
```

من الممكن كتابة جميع التعبيرات المعقدة (التعبيرات التي تحتوي على أكثر من عامل واحد) ببساطة عن طريق وضع المعاملات والعوامل بالترتيب المناسب بحيث تعطي أسبقيتها النسبية الإجابة التي تريدها. ومع ذلك، فإن معظم المبرمجين يكتبون عوامل التشغيل بالترتيب الذي يشعرون أنه أكثر منطقية بالنسبة لهم، ويضيفون أقواسًا للتأكد من أنها منطقية لـ PHP أيضًا. يؤدي الحصول على الأولوية بشكل خاطئ إلى رمز مثل:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

يصعب قراءة هذا الرمز ومن شبه المؤكد أنه لا يفعل ما توقع المبرمج أن يفعله.

تتمثل إحدى الطرق التي يتعامل بها العديد من المبرمجين مع قواعد الأسبقية المعقدة في لغات البرمجة في تقليل الأسبقية إلى قاعدتين:

- ❖ الضرب والقسمة لهما أسبقية أعلى من الجمع والطرح.
- ❖ استخدم الأقواس لأي شيء آخر.

## عمليات ترابطية Operator Associativity

يحدد الترابط الترتيب الذي يتم فيه تقييم العمليات التي لها نفس ترتيب الأسبقية. على سبيل المثال، انظر إلى:

```
2 / 2 * 2
```

عاملي القسمة والضرب لهما نفس الأسبقية، لكن نتيجة التعبير تعتمد على العملية التي نقوم بها أولاً:

```
2 / (2 * 2) // 0.5
(2 / 2) * 2 // 2
```

معاملات القسمة والضرب مترابطة لليسار؛ هذا يعني أنه في حالات الغموض، يتم تقييم العمليات من اليسار إلى اليمين. في هذا المثال، تكون النتيجة الصحيحة هي 2.

## التعبئة الضمنية Implicit Casting

العديد من العمليات لديها توقعات بشأن معاملاتهم - على سبيل المثال، تتطلب عمليات الرياضيات الثنائية عادةً أن يكون كلا المعاملين من نفس النوع. يمكن لمتغيرات PHP تخزين الأعداد الصحيحة وأرقام الفاصلة العائمة والسلاسل والمزيد، وللحفاظ على أكبر قدر ممكن من تفاصيل النوع بعيداً عن المبرمج قدر الإمكان، تحول PHP القيم من نوع إلى آخر حسب الضرورة.

يسمى تحويل قيمة من نوع إلى آخر casting. هذا النوع من التعبئة الضمنية تسمى: التلاعب بالنوع *type juggling* في PHP. يتم عرض قواعد التلاعب بالنوع الذي تقوم به العمليات الحسابية في الجدول 2-4.

الجدول 2-4. قواعد التعبئة الضمنية لعمليات حسابية ثنائية

| نوع المعامل الأول | نوع المعامل الثاني | عملية التحويل |
| --- | --- | --- |
| عددي صحيح | عشري | يتم تحويل العدد الصحيح إلى رقم عشري |
| عددي صحيح | نصي | يتم تحويل النص إلى رقم؛ إذا كانت القيمة بعد التحويل هي رقم عشري، يتم تحويل العدد الصحيح إلى رقم عشري. |
| رقم عشري | نصي | يتم تحويل النص إلى رقم عشري. |

بعض العمليات الأخرى لديها توقعات مختلفة لمعاملاتهم، وبالتالي لديهم قواعد مختلفة. على سبيل المثال، يحول عامل تسلسل السلسلة كلا المعاملين إلى سلاسل قبل ربطهما:

```
3 . 2.74 // gives the string 32.74
```

يمكنك استخدام سلسلة في أي مكان نتوقع PHP رقمًا. من المفترض أن تبدأ السلسلة بعدد صحيح أو رقم عشري. إذا لم يتم العثور على رقم في بداية السلسلة، فإن القيمة الرقمية لتلك السلسلة هي 0. إذا كانت السلسلة تحتوي على نقطة (.) أو حرف كبير أو حرف e صغير، فإن تقييمها رقميًا ينتج عنه رقم عشري. فمثلا:

```
"9 Lives" - 1; // 8 (int)
"3.14 Pies" * 2; // 6.28 (float)
"9. Lives" - 1; // 8 (float / double)
```

```
"1E3 Points of Light" + 1; // 1001 (float)
```

## العمليات الحسابية

العمليات الحسابية هي عمليات ستتعرف عليها من الاستخدام اليومي. معظم العمليات الحسابية ثنائية؛ ومع ذلك، فإن عمليات النفي الحسابي والتأكيد الحسابي أحادية. تتطلب هذه العمليات قيمًا رقمية، ويتم تحويل القيم غير الرقمية إلى قيم رقمية وفقًا للقواعد الموضحة في قسم "عوامل الصب". العوامل الحسابية هي:

**الجمع (+):**

نتيجة عامل الجمع هو مجموع المعاملين.

**الطرح (-):**

نتيجة عامل الطرح هو الفرق بين المعاملين -أي قيمة المعامل الثاني مطروح من الأول-.

**الضرب (*):**

نتيجة عامل الضرب هو حاصل ضرب المعاملين. على سبيل المثال: 3 * 4 هي 12.

**القسمة (/):**

نتيجة عامل القسمة هي حاصل قسمة المعاملين. يمكن أن تعطي قسمة عددين صحيحين عددًا صحيحًا (على سبيل المثال: 2/4) أو عددا عشريا (على سبيل المثال: 2/1).

**باقي القسمة (%):**

يحول عامل المقياس كلا المعاملين إلى أعداد صحيحة ويعيد ما تبقى من قسمة المعامل الأول بواسطة المعامل الثاني. على سبيل المثال: 10٪ 6 تعطي الباقي من 4.

**النفي الحسابي (-)**

يعيد عامل النفي الحسابي المعامل مضروبًا في 1-، مما يؤدي إلى تغيير علامته بشكل فعال. على سبيل المثال: يتم تقييم (4 – 3)- إلى 1. يختلف النفي الحسابي عن عامل الطرح، على الرغم من كتابتهما كعلامة ناقص. النفي الحسابي هو دائمًا أحادي وقبل المعامل. الطرح ثنائي وبين معاملاته.

**التوكيد الحسابي (+)**

يعرض عامل التوكيد الحسابي المعامل مضروبًا في 1+، والذي ليس له أي تأثير. يتم استخدامه فقط كإشارة بصرية للإشارة إلى علامة القيمة. على سبيل المثال: (3 - 4)+ تُقيم بـ -1، تمامًا كما تفعل (3-4).

**الأُس (**)**

يُرجع معامل الأُس نتيجة رفع `$var1` إلى الأس `$var2`.

```
$var1 = 5;
$var2 = 3;
echo $var1 ** $var2; // outputs 125
```

**عامل تسلسل السلسلة**

تعتبر معالجة السلاسل جزءًا أساسيًا من تطبيقات PHP حيث أن PHP لديها عامل تسلسل سلسلة منفصل (.). يلحق عامل التسلسل المعامل الأيمن بالمعامل الأيسر ويعيد السلسلة الناتجة. يتم تحويل المعاملات أولاً إلى سلاسل، إذا لزم الأمر. فمثلا:

```
$n = 5;
$s = 'There were ' . $n . ' ducks.';
// $s is 'There were 5 ducks'
```

عامل التسلسل ذو كفاءة عالية، لأن الكثير من PHP يتلخص في تسلسل السلسلة.

## عوامل الزيادة التلقائية والتناقص التلقائي

في البرمجة، تتمثل إحدى العمليات الأكثر شيوعًا في زيادة أو تقليل قيمة متغير بمقدار واحد. توفر عوامل الزيادة التلقائية الأحادية (++) والتناقص التلقائي (--) اختصارات لهذه العمليات الشائعة. هذه العوامل فريدة من نوعها من حيث أنها تعمل فقط على المتغيرات؛ تغير العمليات قيم معاملاتهم وتعيد قيمة.

هناك طريقتان لاستخدام الزيادة التلقائية أو التناقص التلقائي في التعبيرات. إذا وضعت العامل أمام المعامل، فإنه يُرجع القيمة الجديدة للمعامل (زيادة أو تناقص). إذا وضعت العامل بعد المعامل، فإنه يُرجع القيمة الأصلية للمعامل (قبل الزيادة أو الإنقاص). يسرد الجدول 2-5 العمليات المختلفة.

الجدول 2-5. عمليات الزيادة التلقائية والتناقص التلقائي

| التأثير على المتغير | القيمة المعادة | الاسم | العامل |
| --- | --- | --- | --- |
| زيادة | $var | بعد الزيادة | $var++ |
| زيادة | $var + 1 | قبل الزيادة | ++$var |
| نقصان | $var | بعد النقصان | $var-- |
| نقصان | $var - 1 | قبل النقصان | --$var |

يمكن تطبيق هذه العوامل على السلاسل وكذلك الأرقام. إن زيادة الحرف الأبجدي يحوله إلى الحرف التالي في الأبجدية. كما هو موضح في الجدول 2-6، تؤدي زيادة "z" أو "Z" إلى التفافه مرة أخرى إلى "a" أو "A" وزيادة الحرف السابق بمقدار واحد (أو إدراج حرف "a" أو "A" جديد إذا كان في الأول من السلسلة)، كما لو كانت الأحرف في نظام رقم الأساس 26.

الجدول 2-6. زيادة تلقائية بالحروف

| زيادة هذا | يعطيك هذا |
|---|---|
| "a" | "b" |
| "z" | "aa" |
| "spaz" | "spba" |
| "K9" | "L0" |
| "42" | "43" |

## عوامل المقارنة

كما يوحي اسمها، يقارن عوامل المقارنة المعاملات. تكون النتيجة دائمًا إما `true`، إذا كانت المقارنة صحيحة، و`false` على خلاف ذلك.

يمكن أن تكون معاملات عوامل المقارنة على حد سواء رقمية أو سلسلة أو سلسلة واحدة رقمية وسلسلة واحدة. يتحقق العامل من المصداقية بطرق مختلفة قليلاً بناءً على أنواع وقيم المعاملات، إما باستخدام مقارنات رقمية صارمة أو باستخدام مقارنات معجمية (نصية). يوضح الجدول 2-7 متى يتم استخدام كل نوع من الأنواع.

الجدول 2-7. نوع المقارنة التي يتم إجراؤها بواسطة عوامل المقارنة

| المقارنة | المعامل الثاني | المعامل الأول |
|---|---|---|
| رقمي | ر قم | رقمي |
| رقمي | سلسلة رقمية بالكامل | سلسلة رقمية بالكامل |
| رقمي | رقم | سلسلة رقمية بالكامل |
| معجمي | سلسلة رقمية ليست بالكامل | سلسلة رقمية بالكامل |
| رقمي | رقم | سلسلة رقمية ليست بالكامل |
| معجمي | سلسلة رقمية ليست بالكامل | سلسلة رقمية ليست بالكامل |

شيء واحد مهم يجب ملاحظته هو أنه يتم مقارنة سلسلتين رقميتين كما لو كانتا أرقام. إذا كان لديك سلسلتان تتكونان كلياً من أحرف رقمية وتحتاج إلى مقارنتها معجماً، فاستخدم الدالة `strcmp()`.

**عوامل المقارنة هي:**

**المساواة (==):**

إذا كان كلا المعاملين متساويين، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false.

**التطابق (===):**

إذا كان كلا المعاملين متساويين وكانا من نفس النوع، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false. لاحظ أن هذا العامل لا يقوم بعملية صب النوع الضمني. يكون هذا العامل مفيدًا عندما لا تعرف ما إذا كانت القيم التي تقارنها من النوع نفسه. قد تتضمن المقارنة البسيطة تحويل القيمة. على سبيل المثال، السلاسل "0.0" و "0" غير متساويتين. يقول العامل == إنهم كذلك، لكن === يقول إنهم ليسوا كذلك.

**عدم المساواة (!= أو <>)**

إذا كانت المعاملات غير متساوية، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false.

**غير متطابق (!==)**

إذا كانت المعاملات غير متساوية، أو لم تكن من نفس النوع، فإن هذا العامل يرجع القيمة true؛ وإلا فإنه يرجع false.

**أكبر من (>)**

إذا كان المعامل الأيسر أكبر من المعامل الأيمن، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

**أكبر من أو يساوي (>=)**

إذا كان المعامل الأيسر أكبر من أو يساوي المعامل الأيمن، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

### أقل من (<)

إذا كان المعامل الأيسر أقل من المعامل الأيمن، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

### أصغر من أو يساوي (<=)

إذا كان المعامل الأيسر أقل من المعامل الأيمن أو مساويًا له، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

### سفينة الفضاء (<=>)، والمعروفة أيضًا باسم "Darth Vader's TIE Fighter"

عندما يتساوى المعاملين الأيمن والأيسر، فإن هذا العامل يُرجع 0؛ عندما يكون المعامل الأيسر أقل من المعامل الأيمن، فإنه يُرجع -1؛ وعندما يكون المعامل الأيسر أكبر من المعامل الأيمن، فإنه يعيد 1.

```
$var1 = 5;
$var2 = 65;

echo $var1 <=> $var2 ; // outputs -1
echo $var2 <=> $var1 ; // outputs 1
```

### عامل دمج فارغ *"Null coalescing operator"* (؟؟)

يقوم هذا العامل بتقييم المعامل الأيمن إذا كان المعامل الأيسر NULL؛ خلاف ذلك، يتم تقييمه إلى المعامل الأيسر.

```
$var1 = null;
$var2 = 31;
```

```
echo $var1 ?? $var2 ; //outputs 31
```

## عوامل Bitwise

تعمل عوامل البت على التمثيل الثنائي لمعاملاتها. يتم تحويل كل مُعامل أولاً إلى تمثيل ثنائي للقيمة، كما هو موضح في إدخال عامل النفي على مستوى البت في القائمة التالية. تعمل جميع معاملات البت على الأرقام وكذلك السلاسل، ولكنها تختلف في معالجتها لمعاملات السلسلة ذات الأطوال المختلفة. عوامل تشغيل البت هي:

### النفي على مستوى البت *"Bitwise negation"* (~)

يتغير عامل النفي على مستوى البت من 1s إلى 0s و 0s إلى 1s في التمثيلات الثنائية للمعاملات. يتم تحويل القيم العشرية إلى أعداد صحيحة قبل حدوث العملية. إذا كان المعامل عبارة عن سلسلة، فإن القيمة الناتجة تكون سلسلة بنفس طول الأصل، مع رفض كل حرف في السلسلة.

### أحادي المعامل AND (&)

يقارن العامل AND على مستوى أحادي كل بتة مقابلة في التمثيلات الثنائية للمعاملات. إذا كانت كلتا البتتين 1، فإن البتة المقابلة في النتيجة هي 1؛ خلاف ذلك، فإن البت المقابل هو 0. على سبيل المثال، 0755 و 0671 هو 0651. هذا أسهل قليلاً للفهم إذا نظرنا إلى التمثيل الثنائي. Octal 0755 هو رقم ثنائي 111101101، والثماني 0671 هو رقم ثنائي 110111001. يمكننا بعد ذلك بسهولة رؤية البتات الموجودة في كلا العددين والتوصل إلى الإجابة بصريًا:

```
111101101
& 110111001
 ---------
 110101001
```

الرقم الثنائي 110101001 هو رقم ثماني [2]0651، يمكنك استخدام دالة `bindec()` و `decbin()` و `octdec()` و `decoct()` لتحويل الأرقام ذهابًا وإيابًا عندما تحاول فهم الحساب الثنائي.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية "AND" على مستوى بت بين الحرفين المطابقين في المعاملات. السلسلة الناتجة هي طول أقصر المعاملين؛ يتم تجاهل الأحرف الزائدة في السلسلة الأطول. على سبيل المثال: `"wolf"` و `"cat"` هي `"cad"`.

## أحادي المعامل OR (|)

يقارن العامل OR على مستوى أحادي كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كانت كلتا البتتين 0، فإن البتة الناتجة تكون 0؛ وبخلاف ذلك، يكون البت الناتج هو 1. على سبيل المثال: 020 | 0755 هو 0775.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف هو نتيجة عملية "OR" أحادياً بين الحرفين المتوافقين في المعاملات. السلسلة الناتجة هي طول أطول المعاملين، بينما تكون السلسلة الأقصر مبطنًا في النهاية بـ 0 ثنائي. على سبيل المثال، `"pussy"` | `"cat"` هو `"suwsy"`.

### Bitwise XOR (^)

يقارن عامل البت XOR كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كان أي من البتتين في الزوج، وليس كليهما، هو 1، فإن البتة الناتجة هي 1؛ وبخلاف ذلك، يكون البت الناتج هو 0. على سبيل المثال: 023 ^ 0755 هو 776.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم هذا العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية XOR باتجاه أحادي بين الحرفين المطابقين في المعاملين. إذا كانت الجملتان أطوال مختلفة، فإن السلسلة الناتجة هي طول المعامل الأقصر، ويتم تجاهل الأحرف اللاحقة الإضافية في السلسلة الأطول. على سبيل المثال: "AA" ^ "big drink" تساوي ")#".

### التحول الأيسر (<<)

يقوم عامل الإزاحة لليسار بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر إلى اليسار بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي إلى اليسار إلى إدخال الرقم 0 باعتباره الجزء الأيمن من الرقم ونقل جميع وحدات البت الأخرى إلى اليسار في مكان واحد. على سبيل المثال: 1 << 3 (أو إزاحة 11 ثنائية مكان واحد إلى اليسار) ينتج عنها 6 (ثنائي 110).

لاحظ أن كل مكان على اليسار يتم إزاحة الرقم فيه يؤدي إلى مضاعفة الرقم. نتيجة إزاحة اليسار هي ضرب المعامل الأيسر بمقدار 2 أس المعامل الأيمن.

## التحول الأيمن (>>)

يقوم عامل الإزاحة لليمين بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر والمعامل الأيمن بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي موجب إلى اليمين إلى إدخال الرقم 0 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى المكان الصحيح. يؤدي إزاحة رقم ثنائي سالب إلى اليمين إلى إدخال 1 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى اليمين في مكان واحد. يتم تجاهل البت الموجود في أقصى اليمين. على سبيل المثال: 1 >> 3 (أو ثنائي 1101) إزاحة بت واحد إلى اليمين ينتج 6 (ثنائي 110).

## العمليات المنطقية

توفر لك العوامل المنطقية طرقًا لبناء تعبيرات منطقية معقدة. تعامل العوامل المنطقية معاملاتها كقيم منطقية وتعيد قيمة منطقية. توجد علامات ترقيم وإصدارات باللغة الإنجليزية من عوامل التشغيل (|| و or هو نفس العامل). العوامل المنطقية هي:

## العامل المنطقي (*&&, and*) *AND*

تكون نتيجة عملية AND المنطقية true إذا كان كلا المعاملين true، وإلا فهو false. إذا كانت قيمة المعامل الأول false، فإن العامل المنطقي AND يعرف أن القيمة الناتجة يجب أن تكون false أيضًا، لذلك لا يتم تقييم المعامل الأيمن أبدًا. تسمى هذه العملية بالدائرة القصيرة "*short-circuiting*"، ويستخدمها مصطلح PHP شائع لضمان تقييم جزء من الكود فقط إذا كان هناك شيء صحيح. على سبيل المثال: قد نتصل بقاعدة بيانات فقط إذا كانت بعض العلامات غير false:

```
$result = $flag and mysql_connect();
```

العوامل الـ || و or تختلف فقط في أسبقيتها.

### العامل المنطقي (xor) XOR

تكون نتيجة عملية XOR المنطقية true إذا كان أي من المعاملين true وليس كلاهما، وإلا فهو false.

### عامل النفي المنطقي (!)

يُرجع عامل النفي المنطقي القيمة المنطقية "true" إذا تم تقييم المعامل إلى "false"، وfalse إذا تم تقييم المعامل إلى "true".

## عوامل الصب Casting Operators

على الرغم من أن لغة PHP مكتوبة بشكل ضعيف، إلا أن هناك مناسبات يكون فيها من المفيد اعتبار القيمة كنوع معين. تسمح لك عوامل الصب (int) و (float) و (string) و (bool) و (array) و (object) و (unset) بفرض قيمة في نوع معين. لاستخدام عامل الصب، ضع العامل على يسار المعامل. يسرد الجدول 2-8 عوامل الصب وعوامل التشغيل المترادفة والنوع الذي يغير العامل القيمة إليه.

الجدول 2-8. عوامل صب PHP

| العامل | عوامل مترادفة | تغيير النوع لـ |
|---|---|---|
| (int) | (integer) | صحيح |
| (bool) | (boolean) | منطقي |

| تغيير النوع لـ | عوامل مترادفة | العامل |
| --- | --- | --- |
| عشري | (double), (real) | (float) |
| نصي | | (string) |
| مصفوفة | | (array) |
| كائن | | (object) |
| فارغ | | (unset) |

يؤثر الصب على الطريقة التي يفسر بها العوامل الآخرون القيمة بدلاً من تغيير القيمة في المتغير. على سبيل المثال، الكود:

```
$a = "5";
$b = (int) $a;
```

يعين `$b` القيمة الصحيحة `$a`؛ يبقى `$a` السلسلة "5". لإعطاء قيمة المتغير نفسه، يجب أن تعيد نتيجة التحويل إلى المتغير:

```
$a = "5";
$a = (int) $a; // now $a holds an integer
```

ليس كل عامل صب مفيد. يعطي إرسال مصفوفة إلى نوع رقمي 1 (إذا كانت المصفوفة فارغة، فإنها تعطي 0)، ويؤدي تحويل المصفوفة إلى سلسلة إلى إعطاء "Array" (رؤية هذا في مخرجاتك هو علامة أكيدة على أنك طبعت متغيرًا يحتوي على مجموعة).

يؤدي إرسال كائن إلى مصفوفة إلى إنشاء مصفوفة من الخصائص، وبالتالي تعيين أسماء الخصائص إلى القيم:

```
class Person
{
 var $name = "Fred";
 var $age = 35;
}

$o = new Person;
$a = (array) $o;

print_r($a);
Array ( [name] => Fred [age] => 35)
```

يمكنك تحويل مصفوفة إلى كائن لبناء كائن تتوافق خصائصه مع مفاتيح وقيم المصفوفة. فمثلا:

```
$a = array('name' => "Fred", 'age' => 35, 'wife' =>
"Wilma");
$o = (object) $a;
echo $o->name;
Fred
```

المفاتيح التي لا تعد معرّفات صالحة هي أسماء خصائص غير صالحة ولا يمكن الوصول إليها عندما يتم إرسال مصفوفة إلى كائن، ولكن تتم استعادتها عند إعادة الكائن إلى مصفوفة.

## عوامل التخصيص

يقوم عوامل التخصيص بتخزين أو تحديث القيم في المتغيرات. إن عوامل الزيادة التلقائية والتناقص التلقائي التي رأيناها سابقًا هي عوامل تعيين عالية التخصص - وهنا نرى الأشكال الأكثر عمومية. عامل التخصيص الأساسي هو =، لكننا سنرى أيضًا مجموعات من المهام والعمليات الثنائية، مثل: =+ و =&.

### التخصيص

عامل التخصيص الأساسي (=) يعين قيمة لمتغير. المعامل الأيسر دائمًا متغير. يمكن أن يكون المعامل الأيمن أي تعبير — أي تعبير حرفي بسيط أو متغير أو معقد. يتم تخزين قيمة المعامل الأيمن في المتغير المسمى بواسطة المعامل الأيسر.

نظرًا لأن جميع العوامل مطلوبة لإرجاع قيمة، فإن عامل التعيين يُرجع القيمة المعينة للمتغير. على سبيل المثال: التعبير 5 = a$ لا يعين 5 إلى a$ فحسب، بل يتصرف أيضًا كقيمة 5 إذا تم استخدامه في تعبير أكبر. ضع في اعتبارك التعبيرات التالية:

```
$a = 5;
$b = 10;
$c = ($a = $b);
```

يتم تقييم التعبير $a = $b أولاً بسبب الأقواس. الآن، كلا من $a و $b لهما نفس القيمة، 10. أخيرًا، تم تعيين $c نتيجة التعبير $a = $b، وهي القيمة المخصصة للمعامل الأيسر (في هذه الحالة، $a). عندما ينتهي التعبير الكامل من التقييم، تحتوي جميع المتغيرات الثلاثة على نفس القيمة: 10.

## التخصيص مع عمليات

بالإضافة إلى عامل التخصيص الأساسي، هناك العديد من عوامل التخصيص الذين يعتبرون اختصارًا مناسبًا. تتكون هذه العوامل من عامل ثنائي متبوعًا مباشرة بعلامة يساوي، ويكون تأثيرها هو نفسه تنفيذ العملية باستخدام المعاملات الكاملة، ثم تعيين القيمة الناتجة إلى المعامل الأيسر. عوامل التخصيص هذه هي:

**زائد يساوي (+=)**

يضيف المعامل الأيمن إلى قيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر. `$a += 5` هو نفسه `$a = $a + 5`.

**ناقص يساوي (-=)**

يطرح المعامل الأيمن من قيمة المعامل الأيسر، ثم يعين النتيجة إلى المعامل الأيسر.

**قسمة يساوي (/=)**

يقسم قيمة المعامل الأيسر على المعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

**ضرب يساوي (*=)**

يضرب المعامل الأيمن في قيمة المعامل الأيسر، ثم يخصص النتيجة للمعامل الأيسر.

**المعامل يساوي (%=)**

ينفذ عملية المقياس على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة للمعامل الأيسر.

***Bitwise-XOR-equals* (^=)**

ينفذ XOR باتجاه بت على المعاملين الأيسر والأيمن، ثم يخصص النتيجة للمعامل الأيسر.

***Bitwise-AND-equals* (&=)**

ينفذ "bitwise "AND على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة إلى المعامل الأيسر.

***Bitwise-OR-equals* (|=)**

ينفذ "OR" باتجاه بت على قيمة المعامل الأيسر والمعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

**تسلسل يساوي (=.)**

يربط المعامل الأيمن بقيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر.

## عوامل متنوعة

عوامل تشغيل PHP المتبقية مخصصة لإخماد الأخطاء وتنفيذ أمر خارجي واختيار القيم:

**منع الخطأ (@)**

يمكن لبعض العوامل أو الدوال إنشاء رسائل خطأ. يتم استخدام عامل منع الأخطاء، الذي تمت مناقشته بالكامل في الفصل 17، لمنع إنشاء هذه الرسائل.

**تنفيذ (`...`)**

ينفذ عامل backtick السلسلة الموجودة بين backticks كأمر shell ويعيد الإخراج. فمثلا:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

### الشرط (: ?)

العامل الشرطي، اعتمادًا على الكود الذي تنظر إليه، هو إما المشغل الأكثر استخدامًا أو الأقل استخدامًا. إنه المشغل الثلاثي الوحيد (ثلاثة معاملات)، وبالتالي يُطلق عليه أحيانًا فقط العامل الثلاثي ternary operator.

يقوم العامل الشرطي بتقييم التعبير قبل؟. إذا كان التعبير صحيحًا، فسيقوم العامل بإرجاع قيمة التعبير بين؟ و:، وبخلاف ذلك، يُرجع العامل قيمة التعبير بعد:. على سبيل المثال:

```
<a href="<? echo $url; ?>"><? echo $linktext ? $linktext
: $url; ?></a>
```

إذا كان نص الرابط $url موجودًا في المتغير $linktext، فسيتم استخدامه كنص للرابط، وإلا، يتم عرض عنوان URL نفسه.

### النوع (*instanceof*)

يختبر العامل instanceof ما إذا كان المتغير كائنًا تم إنشاء مثيل له لفئة معينة أو يقوم بتنفيذ واجهة (انظر الفصل 6 لمزيد من المعلومات حول الكائنات والواجهات):

```
$a = new Foo;
$isAFoo = $a instanceof Foo; // true
$isABar = $a instanceof Bar; // false
```

# جمل التحكم في التدفق Flow-Control Statements

يدعم PHP عدداً من تركيبات البرمجة التقليدية للتحكم في تدفق تنفيذ البرنامج.

تسمح الجمل الشرطية، مثل: `if/else` and `switch`، للبرنامج بتنفيذ أجزاء مختلفة من التعليمات البرمجية، أو عدم تنفيذ أي منها على الإطلاق، اعتماداً على بعض الشروط. الحلقات، مثل: while و for، تدعم التنفيذ المتكرر لأجزاء معينة من التعليمات البرمجية.

## if

تحقق جملة if من مصداقية التعبير، وإذا كان التعبير صحيحاً، يتم تقييم الجملة. تبدو جملة if مثل:

```
if (expression)statement
```

لتحديد جملة بديلة لتنفيذه عندما يكون التعبير خاطئاً، استخدم else:

```
if (expression)
 statement
else statement
```

كمثال:

```
if ($user_validated)
 echo "Welcome!";
else
 echo "Access Forbidden!";
```

لتضمين أكثر من جملة واحدة في عبارة if، استخدم block - مجموعة من الجمل محاطة بأقواس معقوفة:

```
if ($user_validated) {
 echo "Welcome!";
 $greeted = 1;
}
else {
 echo "Access Forbidden!";
 exit;
}
```

توفر PHP صيغة أخرى للكُتل في الاختبارات والحلقات. بدلاً من إرفاق كُتلة الجمل بأقواس معقوفة، قم بإنهاء سطر if بنقطتين (:) واستخدم كلمة رئيسية محددة لإنهاء الكُتلة (endif، في هذه الحالة). فمثلا:

```
if ($user_validated):
 echo "Welcome!";
 $greeted = 1;
else:
 echo "Access Forbidden!";
 exit;
endif;
```

الجمل الأخرى الموضحة في هذا الفصل لها أيضًا أنماط بناء جملة بديلة مماثلة (وكلمات رئيسية نهائية)؛ يمكن أن تكون مفيدة إذا كان لديك كتل كبيرة من HTML داخل جملك. فمثلا:

```
<?php if ($user_validated) : ?>
 <table>
```

```
 <tr>
 <td>First Name:</td><td>Sophia</td>
 </tr>
 <tr>
 <td>Last Name:</td><td>Lee</td>
 </tr>
 </table>
<?php else: ?>
 Please log in.
<?php endif ?>
```

لأنه إذا كانت جملة، فيمكنك ربط (تضمين) أكثر من جملة. هذا أيضًا مثال جيد على كيفية استخدام الكتل للمساعدة في الحفاظ على تنظيم الأشياء:

```
if ($good) {
 print("Dandy!");
}
else {
 if ($error) {
 print("Oh, no!");
 }
 else {
 print("I'm ambivalent...");
 }
}
```

هذه السلاسل من جمل if شائعة بدرجة كافية بحيث توفر PHP صياغة أسهل: جملة elseif. على سبيل المثال، يمكن إعادة كتابة الكود السابق على النحو التالي:

```
if ($good) {
 print("Dandy!");
}
elseif ($error) {
 print("Oh, no!");
}
else {
 print("I'm ambivalent...");
}
```

يمكن استخدام المعامل الشرطي الثلاثي (: ?) لتقصير اختبارات الصواب / الخطأ البسيطة. اتخذ موقفًا شائعًا، مثل التحقق لمعرفة ما إذا كان متغير معين صحيحًا وطباعة شيء ما إذا كان كذلك. مع جملة if/else العادية، يبدو الأمر كما يلي:

```
<td><?php if($active) { echo "yes"; } else { echo "no";
} ?></td>
```

مع عامل التشغيل الشرطي الثلاثي، يبدو الأمر كما يلي:

```
<td><?php echo $active ? "yes" : "no"; ?></td>
```

مقارنة بنية الجملة للاثنين:

**if** (*expression*) { *true_statement* } **else** { *false_statement* }

$(expression)\ ?\ true_expression : false_expression$

الاختلاف الرئيسي هنا هو أن العامل الشرطي ليس جملة على الإطلاق. هذا يعني أنه يتم استخدامه في التعبيرات، ونتيجة التعبير الثلاثي الكامل هو في حد ذاته تعبير. في المثال السابق، تكون جملة echo داخل شرط if، بينما عند استخدامها مع العامل الثلاثي، فإنها تسبق التعبير.

switch

قد تحدد قيمة متغير واحد واحدًا من عدد من الخيارات المختلفة (على سبيل المثال، يحتفظ المتغير باسم المستخدم وتريد أن تفعل شيئًا مختلفًا لكل مستخدم). تم تصميم بيان التبديل لهذا الموقف فقط.

تُعطى جملة switch تعبيرًا ويقارن قيمته بجميع الحالات في الـ switch؛ يتم تنفيذ جميع الجمل في حالة المطابقة، حتى أول كلمة رئيسية break تعثر عليها. إذا لم يكن هناك تطابق، وتم تحديد القيمة `default`، فسيتم تنفيذ جميع الجمل التي تتبع الكلمة الأساسية `default`، حتى أول كلمة رئيسية break مصادفة.

على سبيل المثال، افترض أن لديك ما يلي:

```
if ($name == 'ktatroe') {
 // do something
}
else if ($name == 'dawn') {
```

```
 // do something
}
else if ($name == 'petermac') {
 // do something
}
else if ($name == 'bobk') {
 // do something
}
```

يمكنك استبدال هذه الجملة بجملة التبديل التالية:

```
switch($name) {
 case 'ktatroe':
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
}
```

الصيغة البديلة لذلك هي:

```
switch($name):
 case 'ktatroe':
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
endswitch;
```

نظرًا لأنه يتم تنفيذ الجمل من تسمية حالة المطابقة إلى الكلمة الأساسية break التالية، يمكنك دمج العديد من الحالات في *fall-through*. في المثال التالي، تتم طباعة "yes" عندما يكون $name مساويًا لـ sylvie أو bruno:

```
switch ($name) {
 case 'sylvie': // fall-through
 case 'bruno':
 print("yes");
 break;
 default:
 print("no");
```

```
 break;
}
```

يعد التعليق على حقيقة أنك تستخدم حالة سقوط fall-through في التبديل فكرة جيدة، لذلك لا يأتي شخص ما في وقت ما ويضيف `break` معتقدًا أنك قد نسيتها.

يمكنك تحديد عدد اختياري من المستويات للكلمات الرئيسية `break`. بهذه الطريقة، يمكن أن تخرج جملة break من عدة مستويات من جمل التبديل `switch` المتداخلة. يتم عرض مثال على استخدام break بهذه الطريقة في القسم التالي.

while

أبسط شكل من أشكال الحلقة هو جملة while:

```
while (expression)statement
```

إذا تم تقييم التعبير "expression" إلى true، فسيتم تنفيذ الجملة "statement" ثم إعادة تقييم التعبير (إذا كان لا يزال صحيحًا، فسيتم تنفيذ جسم الحلقة مرة أخرى، وهكذا). تخرج الحلقة عندما لا يكون التعبير صحيحًا (أي يتم تقييمه على false).

على سبيل المثال، إليك بعض التعليمات البرمجية التي تضيف الأعداد الصحيحة من 1 إلى 10:

```
$total = 0;
$i = 1;
```

```
while ($i <= 10) {
 $total += $i;
 $i++;
}
```

الصيغة البديلة لـ while لها هذه البنية:

```
while (expr):
 statement;
 more statements ;
endwhile;
```

كمثال:

```
$total = 0;
$i = 1;

while ($i <= 10):
 $total += $i;
 $i++;
endwhile;
```

يمكنك الخروج من حلقة قبل الأوان باستخدام الكلمة الأساسية `break`. في الكود التالي، `$i` لا تصل أبدًا إلى القيمة 6، لأن الحلقة تتوقف بمجرد وصولها إلى 5:

```
$total = 0;
$i = 1;

while ($i <= 10) {
 if ($i == 5) {
 break; // breaks out of the loop
 }

 $total += $i;
 $i++;
}
```

اختياريًا، يمكنك وضع رقم بعد الكلمة الأساسية break يشير إلى عدد مستويات هياكل الحلقة التي يجب الخروج منها. بهذه الطريقة، يمكن أن تندلع عبارة مدفونة عميقاً في حلقات متداخلة من الحلقة الخارجية. فمثلا:

```
$i = 0;
$j = 0;

while ($i < 10) {
 while ($j < 10) {
 if ($j == 5) {
 break 2; // breaks out of two while loops
 }

 $j++;
```

```
 }

 $i++;
}

echo "{$i}, {$j}";
```

**0, 5**

تتخطى جملة continue إلى الاختبار التالي لحالة الحلقة. كما هو الحال مع الكلمة الأساسية break، يمكنك المتابعة عبر عدد اختياري من مستويات بنية الحلقة:

```
while ($i < 10) {
 $i++;

 while ($j < 10) {
 if ($j == 5) {
 continue 2; // continues through two levels
 }

 $j++;
 }
}
```

في هذا الكود، لا يحتوي $j على قيمة أعلى من 5، ولكن $i يمر عبر جميع القيم من 0 إلى 9.

تدعم PHP أيضًا حلقة do/while، والتي تأخذ الشكل التالي:

```
do
 statement
while (expression)
```

استخدم حلقة do/while لضمان تنفيذ جسم الحلقة مرة واحدة على الأقل (المرة الأولى):

```
$total = 0;
$i = 1;

do {
 $total += $i++;
} while ($i <= 10);
```

يمكنك استخدام جمل break و continue في جمل do / while تمامًا كما هو الحال في جملة while العادية.

تُستخدم جملة do/while أحيانًا لكسر كتلة التعليمات البرمجية عند حدوث حالة خطأ. فمثلا:

```
do {
 // do some stuff

 if ($errorCondition) {
 break;
```

```
 }

 // do some other stuff
} while (false);
```

لأن شرط الحلقة خاطئ، يتم تنفيذ الحلقة مرة واحدة فقط، بغض النظر عما يحدث داخل الحلقة. ومع ذلك، في حالة حدوث خطأ، لا يتم تقييم الكود بعد الفاصل.

for

الجملة for مشابهة لجملة while، إلا أنها تضيف عدادات "counter initialization" وتعبيرات عداد "counter manipulation expressions"، وغالبًا ما تكون أقصر وأسهل في القراءة من جمل حلقة while.

إليك حلقة while التي تعد من 0 إلى 9، وطباعة كل رقم:

```
$counter = 0;

while ($counter < 10) {
 echo "Counter is {$counter} <br/>";
 $counter++;
}
```

إليك حلقة for المقابلة والأكثر إيجازًا:

```
for ($counter = 0; $counter < 10; $counter++) {
 echo "Counter is $counter <br/>";
}
```

بنية الجملة for هو:

```
for (start; condition; increment) { statement(s); }
```

يتم تقييم بداية "start" التعبير "expression" مرة واحدة، في بداية جملة for. في كل مرة خلال الحلقة، يتم اختبار شرط "*condition*" التعبير. إذا كان هذا صحيحًا، فسيتم تنفيذ جسم الحلقة؛ إذا كانت خاطئة، تنتهي الحلقة. يتم تقييم زيادة "*increment*" التعبير بعد تشغيل جسم الحلقة.

الصيغة البديلة لجملة for هي:

```
for (expr1; expr2; expr3):
 statement;
 ...;
endfor;
```

يضيف هذا البرنامج الأرقام من 1 إلى 10 باستخدام حلقة for:

```
$total = 0;

for ($i= 1; $i <= 10; $i++) {
 $total += $i;
}
```

إليك نفس الحلقة باستخدام البنية البديلة:

```
$total = 0;

for ($i = 1; $i <= 10; $i++):
 $total += $i;
endfor;
```

يمكنك تحديد عدة تعبيرات لأي من التعبيرات في جملة for عن طريق فصل التعبيرات بفاصلات. فمثلا:

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
 $total += $j;
}
```

يمكنك أيضًا ترك تعبير فارغ، مما يشير إلى أنه لا ينبغي فعل أي شيء لهذه المرحلة. في الشكل الأكثر انحطاطًا، تصبح جملة for حلقة لا نهائية. ربما لا ترغب في تشغيل هذا المثال، لأنه لا يتوقف عن الطباعة:

```
for (;;) {
 echo "Can't stop me!<br />";
}
```

في حلقات for، كما هو الحال في حلقات while، يمكنك استخدام الكلمات الرئيسية break وcontinue لإنهاء الحلقة أو التكرار الحالي.

## foreach

تسمح لك جملة foreach بتكرار العناصر في المصفوفة. تمت مناقشة شكلي جملة foreach بمزيد من التفصيل في الفصل الخامس، حيث نتحدث بمزيد من العمق عن المصفوفات. للتكرار على مصفوفة للوصول إلى القيمة عند كل مفتاح، استخدم:

```
foreach ($array as $current) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $current):
 // ...
endforeach;
```

للتكرار على مصفوفة للوصول إلى كل من المفتاح والقيمة، استخدم:

```
foreach ($array as $key => $value) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $key => $value):
 // ...
endforeach;
```

try...catch

إن بنية try ... catch ليست بنية تحكم في التدفق بقدر ما هي طريقة أكثر رشاقة للتعامل مع أخطاء النظام. على سبيل المثال، إذا كنت تريد التأكد من أن تطبيق الويب الخاص بك لديه اتصال صالح بقاعدة بيانات قبل المتابعة، فيمكنك كتابة كود مثل هذا:

```
try {
 $dbhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);
 doDB_Work($dbhandle); // call function on gaining a connection
 $dbhandle = null; // release handle when done
}
catch (PDOException $error) {
 print "Error!: " . $error->getMessage() . "<br/>";
 die();
}
```

هنا تتم محاولة الاتصال في جزء try من البناء وإذا كان هناك أي أخطاء به، فإن تدفق الكود يقع تلقائيًا في جزء catch، حيث يتم إنشاء مثيل فئة `PDOException` في متغير `$error`. يمكن بعد ذلك عرضها على الشاشة ويمكن أن يفشل الكود "بأمان"، بدلاً من وضع نهاية مفاجئة. يمكنك حتى إعادة التوجيه لمحاولة الاتصال بقاعدة بيانات بديلة، أو الرد على الخطأ بأي طريقة أخرى تريدها داخل جزء catch.

**ملاحظة:**

راجع الفصل 9 لمزيد من الأمثلة عن try ... catch فيما يتعلق بـ PDO (كائنات بيانات PHP "PHP Data Objects") ومعالجة المعاملات.

## الإعلان declare

جملة `declare` تسمح لك بتحديد توجيهات التنفيذ لكتلة من التعليمات البرمجية. بنية جملة declare هو:

```
declare (directive)statement
```

حاليًا، لا يوجد سوى ثلاثة أشكال لـ `declare`: الـ `ticks`، والـ `encoding`، والـ `strict_types`. يمكنك استخدام توجيه `ticks` لتحديد عدد المرات (التي يتم قياسها تقريبيًا في عدد جمل الكود) يتم تسجيل دالة ticks عند استدعاء `register_tick_function()`. فمثلا:

```
register_tick_function("someFunction");

declare(ticks = 3) {
 for($i = 0; $i < 10; $i++) {
 // do something
 }
}
```

في هذا الكود، يتم استدعاء `someFunction()` بعد تنفيذ كل جملة ثالثة داخل الكتلة.

يمكنك استخدام توجيه encoding لتحديد ترميز إخراج نص PHP البرمجي. فمثلا:

```
declare(encoding = "UTF-8");
```

يتم تجاهل هذا النموذج من جملة declare إلا إذا جمعت PHP باستخدام الخيار --enable-zend-multibyte.

أخيرًا، يمكنك استخدام التوجيه strict_types لفرض استخدام أنواع البيانات الصارمة عند تعريف المتغيرات واستخدامها.

exit and return

بمجرد الوصول إليها، تنهي جملة exit تنفيذ البرنامج النصي. ترجع جملة return من دالة أو من البرنامج النصي في المستوى الأعلى من البرنامج.

جملة exit تأخذ قيمة اختيارية. إذا كان هذا رقمًا، فهو يمثل حالة الخروج من العملية. إذا كانت سلسلة، تتم طباعة القيمة قبل إنهاء العملية. الدالة die() هي اسم مستعار لهذا الشكل من جملة exit:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);

if (!$db) {
 die("Could not connect to database");
}
```

يتم كتابة هذا بشكل أكثر شيوعًا على النحو التالي:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
 or die("Could not connect to database");
```

راجع الفصل 3 لمزيد من المعلومات حول استخدام جملة الإرجاع في الوظائف.

goto

تسمح جملة goto للتنفيذ "بالقفز" إلى مكان آخر في البرنامج. يمكنك تحديد نقاط التنفيذ عن طريق إضافة تسمية "label"، وهي عبارة عن معرف متبوع بنقطتين (:). ثم تنتقل بعد ذلك إلى label من موقع آخر في البرنامج النصي عبر جملة goto:

```
for ($i = 0; $i < $count; $i++) {
 // oops, found an error
 if ($error) {
 goto cleanup;
 }
}

cleanup:
// do some cleanup
```

يمكنك فقط الحصول على label في نفس النطاق مثل جملة goto نفسها، ولا يمكنك القفز إلى حلقة أو تبديل. بشكل عام، في أي مكان قد تستخدم فيه goto (أو جملة break، لهذا الأمر)، يمكنك إعادة كتابة الكود ليكون أكثر نظافة بدونه.

# تضمين كود

توفر PHP بنائين لتحميل التعليمات البرمجية و HTML من وحدة نمطية أخرى: `require` و `include`. كلاهما يقوم بتحميل ملف أثناء تشغيل برنامج PHP نصي، ويعملان في الشرطية والحلقات، ويشكو إذا تعذر العثور على الملف الجاري تحميله. توجد الملفات بواسطة مسار ملف مضمن كجزء من التوجيه في استخدام الدالة، أو بناءً على إعداد `include_path` في ملف *php.ini*. يمكن تجاوز طريقة `include_path` بواسطة الدالة `set_include_path()`. إذا فشلت كل هذه الطرق، فإن محاولة PHP الأخيرة هي محاولة العثور على الملف في نفس المجلد مثل البرنامج النصي للاستدعاء. الاختلاف الرئيسي هو أن في `require` إذا طلبت ملف غير موجود يظهر fatal error، بينما محاولة تضمين "`include`" مثل هذا الملف تنتج تحذيرًا ولكنها لا توقف تنفيذ البرنامج النصي.

الاستخدام الشائع لـ `include` هو فصل المحتوى الخاص بالصفحة عن تصميم الموقع العام. تنتقل العناصر الشائعة مثل الرؤوس والتذييلات في ملفات HTML منفصلة، وبعد ذلك تبدو كل صفحة كما يلي:

```
<?php include "header.html"; ?>
content
<?php include "footer.html"; ?>
```

نستخدم التضمين "`include`" لأنه يسمح لـ PHP بمواصلة معالجة الصفحة حتى إذا كان هناك خطأ في ملف (ملفات) تصميم الموقع. الـ `require` أقل تسامحًا وتكون أكثر ملاءمة لتحميل مكتبات التعليمات البرمجية، حيث لا يمكن عرض الصفحة إذا لم يتم تحميل المكتبات. فمثلا:

```
require "codelib.php";
```

```
mysub(); // defined in codelib.php
```

هناك طريقة أكثر فاعلية للتعامل مع الرؤوس والتذييلات وهي تحميل ملف واحد ثم استدعاء الدوال لإنشاء عناصر الموقع الموحدة:

```
<?php require "design.php";
header(); ?>
content
<?php footer();
```

إذا لم تستطع PHP تحليل جزء من الملف المضاف بواسطة التضمين "include" أو الطلب "require"، تتم طباعة تحذير ويستمر التنفيذ. يمكنك كتم صوت التحذير عن طريق تقديم المكالمة مسبقًا باستخدام عامل الصمت (@) - على سبيل المثال،@include.

إذا تم تمكين خيار allow_url_fopen من خلال ملف تكوين PHP، php.ini، فيمكنك تضمين ملفات من موقع بعيد عن طريق توفير عنوان URL بدلاً من مسار محلي بسيط:

```
include "http://www.example.com/codelib.php";
```

إذا كان اسم الملف يبدأ بـ *http://* أو *https://* أو *ftp://*، فسيتم استرداد الملف من موقع بعيد وتحميله.

يمكن تسمية الملفات المضمنة بـ include وrequire بشكل تعسفي. الامتدادات الشائعة هي: *.php*, *.php*5, and *.html*.

**ملاحظة:**

لاحظ أن إحضار ملف ينتهي بـ *php.* عن بُعد من خادم ويب تم تمكين PHP له يؤدي إلى جلب *output* نص PHP البرمجي هذا — إنه ينفذ كود PHP في هذا الملف.

إذا كان أحد البرامج يستخدم `include` أو `require` لتضمين نفس الملف مرتين (تم القيام به عن طريق الخطأ في حلقة، على سبيل المثال)، يتم تحميل الملف وتشغيل الكود، أو طباعة HTML مرتين. يمكن أن يؤدي هذا إلى أخطاء حول إعادة تعريف الوظائف، أو إرسال نسخ متعددة من الرؤوس أو HTML. لمنع حدوث هذه الأخطاء، استخدم التركيبات `include_once` و `require_once`. يتصرفون بنفس سلوك `include` و`require` المرة الأولى التي يتم فيها تحميل الملف، لكنهم يتجاهلون بهدوء المحاولات اللاحقة لتحميل نفس الملف. على سبيل المثال، تحتاج العديد من عناصر الصفحة، كل واحدة مخزنة في ملفات منفصلة، إلى معرفة تفضيلات المستخدم الحالي. يجب أن تقوم مكتبات العناصر بتحميل مكتبة تفضيلات المستخدم مع `require_once`. يمكن لمصمم الصفحة بعد ذلك تضمين عنصر صفحة دون القلق بشأن ما إذا كان قد تم تحميل كود تفضيل المستخدم بالفعل.

يتم استيراد الكود الموجود في ملف مضمن في النطاق الذي يتم فيه العثور على جملة `include`، لذلك يمكن للكود المضمن رؤية متغيرات التعليمات البرمجية الخاصة بك وتغييرها. يمكن أن يكون هذا مفيدًا - على سبيل المثال، قد تخزن مكتبة تتبع المستخدم اسم المستخدم الحالي في المتغير `$user` العام:

```
// main page
include "userprefs.php";
echo "Hello, {$user}.";
```

يمكن أن تكون قدرة المكتبات على رؤية المتغيرات الخاصة بك وتغييرها مشكلة أيضًا. يجب أن تعرف كل متغير عام تستخدمه مكتبة للتأكد من أنك لا تحاول عن طريق الخطأ استخدام أحدها لأغراضك الخاصة، وبالتالي الكتابة فوق قيمة المكتبة وتعطيل طريقة عملها.

إذا كانت بنية include أو require في دالة، فإن المتغيرات في الملف المضمن تصبح متغيرات نطاق الدالة لتلك الدالة.

نظرًا لأن الكلمات الرئيسية include و require، وليست جمل حقيقية، فيجب عليك دائمًا تضمينها بأقواس معقوفة في الجمل الشرطية والحلقة:

```
for ($i = 0; $i < 10; $i++) {
 include "repeated_element.html";
}
```

استخدم دالة get_included_files() لمعرفة الملفات التي تم تضمينها أو طلبها في النص البرمجي. تقوم بإرجاع مصفوفة تحتوي على أسماء ملفات مسار النظام الكاملة لكل ملف مضمن أو مطلوب. لا يتم تضمين الملفات التي لم يتم تحليلها في هذه المصفوفة.

# تضمين PHP في صفحات الويب

على الرغم من أنه من الممكن كتابة وتشغيل برامج PHP مستقلة، فإن معظم أكواد PHP مضمنة في ملفات HTML أو XML. هذا، بعد كل شيء، هو سبب إنشائها في المقام الأول. تتضمن معالجة مثل هذه المستندات استبدال كل جزء من كود مصدر PHP بالمخرجات التي تنتجها عند تنفيذها.

نظراً لأن الملف الفردي عادةً ما يحتوي على PHP وكود مصدر غير PHP، فنحن بحاجة إلى طريقة لتحديد مناطق كود PHP المراد تنفيذه. توفر PHP أربع طرق مختلفة للقيام بذلك.

كما سترى، الطريقة الأولى والمفضلة تبدو مثل XML. الطريقة الثانية تبدو مثل SGML. الطريقة الثالثة تعتمد على أوسمة ASP. الطريقة الرابعة تستخدم وسم HTML القياسية `<script>`، هذا يجعل من السهل تحرير الصفحات باستخدام PHP الممكّن باستخدام محرر HTML عادي.

## نمط قياسي (XML)

نظراً لظهور لغة التوصيف القابلة للتوسيع "eXtensible Markup Language" (XML) وترحيل HTML إلى لغة (XHTML) XML، فإن الأسلوب المفضل حالياً لتضمين PHP يستخدم علامات متوافقة مع XML للإشارة إلى تعليمات PHP.

كان من السهل إنشاء أوسمة لتعيين أوامر PHP في XML، لأن XML يسمح بتعريف الأوسمة الجديدة. لاستخدام هذا النمط، قم بإحاطة كود PHP بـ `<?php` and `?>`. يتم تفسير كل شيء بين هذه الأوسمة على

أنه PHP، ولا يتم تفسير أي شيء خارج الأوسمة. على الرغم من أنه ليس من الضروري تضمين مسافات بين العلامات والنص المرفق، فإن القيام بذلك يحسن إمكانية القراءة. على سبيل المثال، لجعل PHP تطبع "Hello, world,"، يمكنك إدراج السطر التالي في صفحة الويب:

```
<?php echo "Hello, world"; ?>
```

الفاصلة المنقوطة اللاحقة في الجملة اختيارية؛ لأن نهاية الكتلة تفرض أيضًا نهاية الجملة. مضمن في ملف HTML كامل، يبدو هذا كما يلي:

```
<!doctype html>
<html>
<head>
 <title>This is my first PHP program!</title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 <?php echo "Hello, world"; ?><br />
 How cool is that?
</p>
</body>

</html>
```

بالطبع، هذا ليس مثيرًا للغاية - كان بإمكاننا فعل ذلك بدون PHP. تأتي القيمة الحقيقية لـ PHP عندما نضع معلومات حيوية من مصادر مثل قواعد البيانات وقيم النموذج في صفحة الويب. هذا لفصل لاحق، مع ذلك. دعنا نعود إلى مثالنا "Hello, world". عندما يزور مستخدم هذه الصفحة ويعرض مصدرها، يبدو الأمر كما يلي:

```
<!doctype html>
<html>
<head>
 <title> This is my first PHP program! </title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 Hello, world!<br />
 How cool is that?
</p>
</body>

</html>
```

لاحظ أنه لا يوجد أي أثر لكود مصدر PHP من الملف الأصلي. يرى المستخدم فقط ناتجها.

لاحظ أيضًا أننا قمنا بالتبديل بين PHP و non-PHP، كل ذلك في مساحة سطر واحد. يمكن وضع تعليمات PHP في أي مكان في الملف، حتى ضمن أوسمة HTML الصالحة. فمثلا:

```
<input type="text" name="first_name" value="<?php echo
"Peter"; ?>" />
```

عندما ينتهي PHP من هذا النص، سيقرأ:

```
<input type="text" name="first_name" value="Peter" />
```

لا يجب أن يكون كود PHP داخل علامات الفتح والإغلاق على نفس السطر. إذا كانت علامة إغلاق تعليمة PHP هي آخر شيء في السطر، فسيتم أيضًا إزالة فاصل السطر الذي يلي علامة الإغلاق. وبالتالي، يمكننا استبدال تعليمات PHP في مثال "Hello, world" بـ:

```
<?php
echo "Hello, world"; ?>
<br />
```

مع عدم وجود تغيير في HTML الناتج.

نمط SGML

يأتي نمط آخر لتضمين PHP من أوسمة معالجة تعليمات SGML. لاستخدام هذه الطريقة، ضع PHP داخل <? و?>. إليك مثال "Hello, world" مرة أخرى:

```
<? echo "Hello, world"; ?>
```

هذا النمط، المعروف باسم الأوسمة القصيرة *short tags*، مغلق افتراضياً. يمكنك تشغيل دعم الأوسمة القصيرة عن طريق إنشاء PHP باستخدام الخيار `--enable-short-tags`، أو تمكين

short_open_tag في ملف تكوين PHP. هذا غير محبذ لأنه يعتمد على حالة هذا الإعداد، إذا قمت بتصدير الكود الخاص بك إلى نظام أساسي آخر، فقد يعمل وقد لا يعمل.

وسم echo قصير، <?= ... ?>، متاحة بغض النظر عن توفر الأوسمة القصيرة.

### echo المحتوى مباشرة

ربما تكون العملية الفردية الأكثر شيوعًا داخل تطبيق PHP هي عرض البيانات للمستخدم. في سياق تطبيق الويب، يعني هذا إدخال معلومات مستند HTML التي ستصبح HTML عند عرضها من قبل المستخدم.

لتبسيط هذه العملية، توفر PHP إصدارًا خاصًا من أوسمة SGML التي تأخذ القيمة تلقائيًا داخل الوسم وتدرجها في صفحة HTML. لاستخدام هذه الميزة، أضف علامة يساوي (=) إلى وسم الفتح. باستخدام هذه التقنية، يمكننا إعادة كتابة مثال النموذج الخاص بنا على النحو التالي:

```
<input type="text" name="first_name" value="<?= "Dawn";
?>">
```

# مالتالي

الآن بعد أن أصبح لديك أساسيات اللغة تحت حزامك - فهم أساسي لماهية المتغيرات وكيفية تسميتها وأنواع البيانات وكيف يعمل التحكم في تدفق التعليمات البرمجية - سننتقل إلى بعض التفاصيل الدقيقة للغة PHP . سنغطي بعد ذلك ثلاثة موضوعات مهمة جدًا لـ PHP بحيث يكون لكل منها فصوله المخصصة: كيفية تحديد الدوال (الفصل 3)، والتعامل مع السلاسل (الفصل 4)، وإدارة المصفوفات (الفصل 5).

---

(1) إنها في الواقع 3 إذا كنت تنظر إلى عدد المراجع من C API، ولكن لأغراض هذا الشرح ومن منظور مساحة المستخدم ، فمن الأسهل التفكير فيه على أنه 2.

(2) إليك نصيحة: قسّم الرقم الثنائي إلى ثلاث مجموعات - 6 عبارة عن ثنائي 110، و 5 هو ثنائي 101، و 1 هو ثنائي 001؛ وبالتالي، 0651 هو 110101001.

# الفصل الثالث: الدوال

**الدالة:** عبارة عن كتلة مسماة من التعليمات البرمجية تؤدي مهمة محددة، وربما تعمل على مجموعة من القيم المعطاة لها، تعرف أيضًا باسم المعلمات "*parameters*"، وربما تعيد قيمة واحدة أو مجموعة من القيم في مصفوفة. توفر الدوال وقت الترجمة - بغض النظر عن عدد المرات التي تستدعى فيها، يتم تجميع الدوال مرة واحدة فقط للصفحة. كما أنها تعمل على تحسين الموثوقية من خلال السماح لك بإصلاح أي أخطاء في مكان واحد بدلاً من أي مكان تقوم فيه بمهمة ما، كما تعمل على تحسين إمكانية القراءة عن طريق عزل التعليمات البرمجية التي تؤدي مهام محددة.

يقدم هذا الفصل بناء جملة استدعاءات الدوال وتعريفات الدوال ويناقش كيفية إدارة المتغيرات في الدوال وتمرير القيم إليها (بما في ذلك التمرير بالقيمة والمرجع التمريري "including pass-by-value and pass-by-reference"). كما أنه يغطي الدوال المتغيرة والدوال المجهولة.

## استدعاء الدالة

يمكن دمج الدوال في برنامج PHP (أو من خلال كونها في ملحق، مضمنة بشكل فعال) أو معرفة من قبل المستخدم. بغض النظر عن مصدرها، يتم تقييم جميع الدوال بنفس الطريقة:

$someValue = *function_name*( [ *parameter, ...* ] );

يختلف عدد المعلمات التي تتطلبها الدالة من دالة إلى أخرى (وكما سنرى لاحقًا، قد تختلف أيضًا لنفس الدالة). قد تكون المعلمات التي يتم توفيرها للدالة أي تعبير صالح ويجب أن تكون بالترتيب المحدد الذي

تتوقعه الدالة. إذا تم إعطاء المعلمات خارج الترتيب، فقد يستمر تشغيل الدالة بواسطة fluke، ولكنها في الأساس حالة "garbage in = garbage out". ستخبرك وثائق الدالة بالمعلمات التي تتوقعها الدالة والقيمة (القيم) التي يمكنك توقع إرجاعها.

فيما يلي بعض الأمثلة على الدوال:

```
// strlen() هي دالة مضمنة ترجع طول السلسلة //
$length = strlen("PHP"); // $length الان 3
// sin() و asin() هي دوال رياضية الجيب والقوس //
$result = sin(asin(1)); // $result is the sine of arcsin(1), or 1.0

// unlink() تحذف ملف
$result = unlink("functions.txt");
// $result = true or false depending on success or failure
```

في المثال الأول، نعطي المدخل "PHP" للدالة `strlen()` والتي تعطينا عدد الأحرف في السلسلة المتوفرة. في هذه الحالة، تقوم بإرجاع 3، والتي تم تخصيصها للمتغير `$length`. هذه هي الطريقة الأبسط والأكثر شيوعًا لاستخدام دالة.

المثال الثاني يمرر نتيجة `asin(1)` إلى دالة `sin()`. نظراً لأن الدالتين الجيب والقوس هي انعكاسات، فإن أخذ جيب القوس لأي قيمة سيعيد دائمًا نفس القيمة. نرى هنا أنه يمكن استدعاء دالة داخل دالة أخرى. يتم إرسال القيمة المرجعة للإستدعاء الداخلي إلى الدالة الخارجية قبل إرجاع النتيجة الإجمالية وتخزينها في متغير `$result`

في المثال الأخير، نعطي اسم ملف لدالة `unlink()`، التي تحاول حذف الملف. مثل العديد من الدوال، فإنها ترجع false عندما تفشل. يتيح لك ذلك استخدام دالة مضمنة أخرى، وهي `die()` وخاصية قصر الدائرة "short-circuiting" لعوامل المنطق. وبالتالي، يمكن إعادة كتابة هذا المثال على النحو التالي:

```
$result = unlink("functions.txt") or die("Operation
failed!");
```

على عكس المثالين الآخرين، تؤثر دالة `unlink()` على شيء خارج المعلمات المعطاة لها. في هذه الحالة، يقوم بحذف ملف من نظام الملفات. يجب توثيق كل هذه الآثار الجانبية للدالة بعناية ودراستها بعناية.

تحتوي PHP على مجموعة كبيرة من الدوال المحددة بالفعل لتستخدمها في برامجك. تؤدي هذه الملحقات مهام مثل الوصول إلى قواعد البيانات وإنشاء الرسومات وقراءة ملفات XML وكتابتها والاستيلاء على الملفات من الأنظمة البعيدة وغير ذلك.

**ملاحظة:**

لا تُرجع جميع الدوال قيمة. يمكنها تنفيذ إجراء مثل إرسال بريد إلكتروني ثم إعادة سلوك التحكم إلى استدعاء الكود، بعد إكمال مهمتها، ليس لديهم ما "يقولونه "say"".

# تعريف الدالة

لتعريف دالة، استخدم الصيغة التالية:

```
function [&] function_name([parameter[, ...]])
{
 قائمة الجمل
}
```

يمكن أن تتضمن قائمة الجمل HTML. يمكنك تعريف دالة PHP لا تحتوي على أي كود PHP. على سبيل المثال، تعطي دالة ()column اسمًا قصيرًا مناسبًا لكود HTML التي قد تكون مطلوبة عدة مرات في جميع أنحاء الصفحة:

```
<?php function column()
{ ?>
 </td><td>
<?php }
```

يمكن أن يكون اسم الدالة أي سلسلة تبدأ بحرف أو شرطة سفلية متبوعة بصفر أو أكثر من الأحرف والشرطات السفلية والأرقام. أسماء الدالة غير حساسة لحالة الأحرف، وهذا يعني أنه يمكنك استدعاء دالة ()sin كـ sin(1), SIN(1), SiN(1) وهكذا، لأن كل هذه الأسماء تشير إلى نفس الدالة. حسب الاصطلاح، يتم استدعاء دوال PHP المدمجة بأحرف صغيرة.

عادةً ما ترجع الدالات بعض القيمة. لإرجاع قيمة من دالة، استخدم جملة return ضع return *expr* داخل دالتك. عند مصادفة جملة return أثناء التنفيذ، يعود عنصر التحكم إلى جملة الاستدعاء، وسيتم إرجاع النتائج المقيمة لـ expr كقيمة للدالة. يمكنك تضمين أي عدد من جمل الإرجاع في دالة (على سبيل المثال، إذا كان لديك جملة switch لتحديد أي من القيم العديدة التي تريد إرجاعها).

دعونا نلقي نظرة على دالة بسيطة. يأخذ المثال 3-1 سلسلتين، يسلسلهما، ثم يُرجع النتيجة (في هذه الحالة، أنشأنا مكافئًا أبطأ قليلاً لعامل التسلسل، لكننا تحمل معنا من أجل المثال).

مثال 3-1. تسلسل السلسلة

```
function strcat($left, $right)
{
 $combinedString = $left . $right;

 return $combinedString;
}
```

تأخذ الدالة مدخلين، $left و $right. باستخدام العامل التسلسلي، تُنشئ الدالة سلسلة مجمعة في المتغير $combinedString. أخيرًا، من أجل جعل الدالة لها قيمة عند تقييمها باستخدام مدخلاتنا، نعيد القيمة $combinedString.

نظرًا لأن جملة return يمكن أن تقبل أي تعبير "expression"، حتى التعبيرات المعقدة، يمكننا تبسيط البرنامج كما هو موضح هنا:

```
function strcat($left, $right)
{
 return $left . $right;
}
```

إذا وضعنا دالة في صفحة php يمكننا استدعائها من أي مكان في هذه الصفحة. ألق نظرة على هذا المثال:

المثال 3-2. استخدام دالة الربط

```
<?php
function strcat($left, $right)
{
 return $left . $right;
}
$first = "This is a ";
$second = " complete sentence!";

echo strcat($first, $second);
```

عندما يتم عرض هذه الصفحة، يتم عرض الجملة الكاملة.

في هذا المثال التالي، تأخذ الدالة عددًا صحيحًا، وتضاعفها عن طريق إزاحة القيمة الأصلية، وتعيد النتيجة:

```
function doubler($value)
{
```

```
 return $value << 1;
}
```

بمجرد تعريف الدالة، يمكنك استخدامها في أي مكان على الصفحة. فمثلا:

```
<?php echo "A pair of 13s is " . doubler(13); ?>
```

يمكنك تداخل تعريفات الدوال، ولكن بتأثير محدود. لا تحد التعريفات المتداخلة من رؤية الدالة المحددة داخليًا، والتي يمكن استدعاؤها من أي مكان في برنامجك. لا تحصل الدالة الداخلية تلقائيًا على مدخلات الدالة الخارجية. وأخيرًا، لا يمكن استدعاء الدالة الداخلية حتى يتم استدعاء الدالة الخارجية، ولا يمكن أيضًا استدعائها من الكود التي تم تحليلها بعد الدالة الخارجية:

```
function outer ($a)
{
 function inner ($b)
 {
 echo "there $b";
 }

 echo "$a, hello ";
}

// outputs "well, hello there reader"
outer("well");
inner("reader");
```

# نطاق متغير Variable Scope

إذا كنت لا تستخدم الدوال، فيمكن استخدام أي متغير تقوم بإنشائه في أي مكان بالصفحة. مع الدوال، هذا ليس صحيحًا دائمًا. تحتفظ الدوال بمجموعات المتغيرات الخاصة بها والتي تختلف عن تلك الخاصة بالصفحة وعن الدوال الأخرى.

لا يمكن الوصول إلى المتغيرات المحددة في دالة، بما في ذلك معلماتها، خارج الدالة، وافتراضيًا، لا يمكن الوصول إلى المتغيرات المحددة خارج الدالة داخل الدالة. يوضح المثال التالي هذا:

```
$a = 3;

function foo()
{
 $a += 2;
}

foo();
echo $a;
```

المتغير $a داخل الدالة foo() هو متغير مختلف عن المتغير $a خارج الدالة؛ على الرغم من أن foo() يستخدم عامل add-and-assign، فإن قيمة $a الخارجية تبقى 3 طوال عمر الصفحة. داخل الدالة، قيمة $a هي 2.

كما ناقشنا في الفصل الثاني، فإن المدى الذي يمكن أن يُرى فيه المتغير في البرنامج يسمى نطاق المتغير. المتغيرات التي تم إنشاؤها داخل دالة موجودة داخل نطاق الدالة (أي لها نطاق على مستوى الدالة). المتغيرات التي تم إنشاؤها خارج الدوال والكائنات لها نطاق عالمي وتوجد في أي مكان خارج تلك الدوال والكائنات. بعض المتغيرات التي توفرها PHP لها نطاق دالة ونطاق عالمي (يشار إليها غالباً باسم المتغيرات العالمية الفائقة *super-global variables*).

للوهلة الأولى، حتى المبرمج المتمرس قد يعتقد أنه في المثال السابق سيكون $a 5 بحلول وقت الوصول إلى جملة echo، لذلك ضع ذلك في الاعتبار عند اختيار أسماء لمتغيراتك.

### المتغيرات العالمية

إذا كنت تريد الوصول إلى متغير في النطاق العام من داخل دالة، يمكنك استخدام الكلمة الأساسية global. تركيبها هو:

```
global var1, var2, ... ;
```

بتغيير المثال السابق لتضمين الكلمة الرئيسية global، نحصل على:

```
$a = 3;

function foo()
{
 global $a;

 $a += 2;
}
```

```
foo();
echo $a;
```

بدلاً من إنشاء متغير جديد يسمى a$ بنطاق على مستوى الدالة، PHP تستخدم global $a داخل الدالة. الآن، عندما يتم عرض قيمة a$، ستكون 5.

يجب عليك تضمين الكلمة الأساسية global في دالة قبل أي استخدامات للمتغير العام أو المتغيرات التي تريد الوصول إليها. نظرًا لأنه يتم الإعلان عنها قبل جسم الدالة، لا يمكن أبدًا أن تكون معلمات الدالة متغيرات عامة.

يُعادل استخدام global إنشاء مرجع للمتغير في المتغير GLOBALS$. أي أن كلا الإعلانين التاليين ينشئان متغيرًا في نطاق الدالة يمثل مرجعًا لنفس قيمة المتغير var$ في النطاق العام:

```
global $var;
$var = & $GLOBALS['var'];
```

## المتغيرات الثابتة

مثل لغة C، تدعم PHP تعريف متغيرات الدالة بأنها ثابتة "*static*". يحتفظ المتغير الثابت بقيمته بين جميع استدعاءات الدالة ويتم تهيئته أثناء تنفيذ البرنامج النصي فقط في المرة الأولى التي يتم فيها استدعاء الدالة. استخدم الكلمة الأساسية *static* عند أول استخدام لمتغير دالة للإعلان عن أنها ثابتة. عادةً ما يخصص أول استخدام لمتغير ثابت قيمة أولية:

```
static var [= value][, ... ];
```

في المثال 3-3، يتم زيادة المتغير $count بمقدار واحد في كل مرة يتم استدعاء الدالة.

المثال 3-3. عداد متغير ثابت

```
<?php
function counter()
{
 static $count = 0;

 return $count++;
}

for ($i = 1; $i <= 5; $i++) {
 print counter();
}
```

عندما يتم استدعاء الدالة لأول مرة، يتم تعيين قيمة 0 للمتغير الثابت $count. يتم إرجاع القيمة وزيادة $count. عندما تنتهي الدالة، لا يتم إتلاف $count كمتغير غير ثابت، وتظل قيمتها كما هي حتى يتم استدعاء counter() المرة التالية. تعرض حلقة for الأرقام من 0 إلى 4.

# معلمات الدالة

يمكن أن نتوقع الدوال عددًا تعسفيًا من المدخلات، التي تم تحديدها بواسطة تعريف الدالة. هناك طريقتان مختلفتان لتمرير المعلمات إلى دالة. الأول والأكثر شيوعًا هو القيمة ”value“. والثاني بالإشارة ”reference“.

## تمرير المعلمات بالقيمة

في معظم الحالات، تقوم بتمرير المعلمات حسب القيمة. المدخل هو أي تعبير صالح. يتم تقييم هذا التعبير، ويتم تعيين القيمة الناتجة إلى المتغير المناسب في الدالة. في جميع الأمثلة حتى الآن، مررنا المدخلات بالقيمة.

## تمرير المعلمات حسب المرجع

يسمح لك التمرير حسب المرجع بتجاوز قواعد تحديد النطاق العادية وإعطاء الدالة وصولاً مباشرًا إلى المتغير. ليتم تمريرها عن طريق المرجع، يجب أن يكون المدخل متغيرًا، أنت تشير إلى أن مدخل معين للدالة سيتم تمريره عن طريق المرجع عن طريق اسم المتغير في قائمة المعلمات بعلامة العطف (&). مثال 3-4 يعيد النظر في دالة ()doubler مع تغيير طفيف.

مثال 3-4. إعادة ()doubler

```
<?php
function doubler(&$value)
{
```

```
 $value = $value << 1;
}

$a = 3;
doubler($a);

echo $a;
```

نظرًا لأن معلمة $value للدالة يتم تمريرها من خلال المرجع، فإن القيمة الفعلية لـ $a، بدلاً من نسخة من تلك القيمة، يتم تعديلها بواسطة الدالة. في السابق، كان علينا إعادة القيمة المضاعفة، لكننا الآن نغير متغير المتصل ليكون القيمة المضاعفة.

هذا مكان آخر حيث يكون للدالة آثار جانبية: بما أننا مررنا المتغير $a إلى doubler() حسب المرجع، فإن قيمة $a تكون تحت رحمة الدالة. في هذه الحالة، يقوم doubler() بتعيين قيمة جديدة له.

يمكن فقط توفير المتغيرات - وليس الثوابت "constants" - للمعلمات المعلنة على أنها تمريرة من خلال المرجع. وبالتالي ، إذا قمنا بتضمين الجملة <?php echo doubler(7); ?> في المثال السابق، سيصدر خطأ. ومع ذلك، يمكنك تعيين قيمة افتراضية للمعلمات التي تم تمريرها بواسطة المرجع (بنفس الطريقة التي توفر بها القيم الافتراضية للمعلمات التي تم تمريرها بواسطة القيمة).

حتى في الحالات التي لا تؤثر فيها دالتك على القيمة المحددة، قد ترغب في تمرير المعلمة عن طريق المرجع. عند تمرير القيمة، يجب أن تنسخ PHP القيمة. يمكن أن تكون هذه عملية مكلفة خاصة بالنسبة للسلاسل النصية والكائنات الكبيرة. التمرير حسب المرجع يزيل الحاجة إلى نسخ القيمة.

## المعلمات الافتراضية

في بعض الأحيان قد تحتاج الدالة إلى قبول معلمة معينة. على سبيل المثال، عند استدعاء دالة للحصول على التفضيلات لموقع ما، قد تأخذ الدالة معلمة باسم التفضيل الذي تريد استرداده. بدلاً من استخدام بعض الكلمات الأساسية الخاصة لتعيين أنك تريد استرداد جميع التفضيلات، لا يمكنك ببساطة تقديم أي مدخل. يعمل هذا السلوك باستخدام المدخلات الافتراضية.

لتحديد معلمة افتراضية، قم بتعيين قيمة المعلمة في إعلان الدالة. لا يمكن أن تكون القيمة المعينة للمعامل كقيمة افتراضية تعبيرًا معقدًا، فقط قيمة عددية:

```
function getPreferences($whichPreference = 'all')
{
 // if $whichPreference is "all", return all prefs;
 // otherwise, get the specific preference requested...
}
```

عند استدعاء ()getPreferences، يمكنك اختيار تقديم مدخل. إذا قمت بذلك، فإنها تُرجع التفضيل المطابق للسلسلة التي تقدمها لها؛ إذا لم يكن كذلك، فإنه يقوم بإرجاع جميع التفضيلات.

**ملاحظة:**

قد تحتوي الدالة على أي عدد من المعلمات ذات القيم الافتراضية. ومع ذلك، يجب إدراج هذه المعلمات الافتراضية بعد كافة المعلمات التي لا تحتوي على قيم افتراضية.

## معلمات المتغير

قد تتطلب الدالة عددًا متغيرًا من المدخلات. على سبيل المثال، قد يُرجع مثال getPreferences() في القسم السابق التفضيلات لأي عدد من الأسماء، بدلاً من اسم واحد فقط. للإعلان عن دالة بعدد متغير من المدخلات، اترك كتلة المعامل بالكامل:

```
function getPreferences()
{
 // some code
}
```

توفر PHP ثلاث دوال يمكنك استخدامها في الدالة لاسترداد المعلمات التي تم تمريرها إليها. يرجع func_get_args () مصفوفة من جميع المعلمات المتوفرة للدالة ؛ تُرجع الدالة func_get_args() عدد المعلمات المتوفرة للدالة؛ ويرجع func_get_arg () مدخل معين من المعلمات. فمثلا:

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(argument_number);
```

في المثال 3-5، تأخذ الدالة ()count_list أي عدد من المدخلات. يقوم بعمل حلقات فوق تلك المدخلات وإرجاع إجمالي جميع القيم. إذا لم يتم إعطاء أي معلمات، فإنها ترجع false.

مثال 3-5. عداد المدخلات

```
<?php
function countList()
{
 if (func_num_args() == 0) {
 return false;
 }
 else {
 $count = 0;

 for ($i = 0; $i < func_num_args(); $i++) {
 $count += func_get_arg($i);
 }

 return $count;
 }
}

echo countList(1, 5, 9); // outputs "15"
```

لا يمكن استخدام نتيجة أي من هذه الدوال مباشرة كمعامل لدالة أخرى. بدلاً من ذلك، يجب عليك أولاً تعيين متغير إلى نتيجة الدالة، ثم استخدامه في استدعاء الدالة. لن يعمل التعبير التالي:

```
foo(func_num_args());
```

بدلاً من ذلك، استخدم:

```
$count = func_num_args();
foo($count);
```

## معلمات مفقودة

تتيح لك PHP أن تكون كسولًا كما تريد - عند استدعاء دالة، يمكنك تمرير أي عدد من المدخلات إلى الدالة. تظل أي معلمات تتوقعها الدالة ولم يتم تمريرها بدون تعيين، ويتم إصدار تحذير لكل منها:

```
function takesTwo($a, $b)
{
 if (isset($a)) {
 echo " a is set\n";
 }

 if (isset($b)) {
 echo " b is set\n";
 }
}

 echo "With two arguments:\n";
takesTwo(1, 2);

echo "With one argument:\n";
```

```
takesTwo(1);
With two arguments:
 a is set
 b is set
With one argument:
Warning: Missing argument 2 for takes_two()
 in /path/to/script.php on line 6
a is set
```

## اكتب تلميح

عند تعريف دالة، يمكنك إضافة تلميح من النوع - أي يمكنك أن تطلب أن تكون المعلمة مثيلًا لفئة معينة (بما في ذلك مثيلات الفئات التي تمد هذه الفئة)، ومثيلًا لفئة تنفذ واجهة معينة، مجموعة أو قابلة للاستدعاء. لإضافة تلميح نوع إلى معلمة، قم بتضمين اسم الفئة أو array أو callable قبل اسم المتغير في قائمة معلمات الدالة. فمثلا:

```
class Entertainment {}

class Clown extends Entertainment {}

class Job {}

function handleEntertainment(Entertainment $a, callable
$callback = NULL)
{
 echo "Handling " . get_class($a) . " fun\n";
```

```
 if ($callback !== NULL) {
 $callback();
  }
}

$callback = function()
{
 // do something
};

handleEntertainment(new Clown); // works
handleEntertainment(new Job, $callback); // runtime error
```

يجب أن تكون المعلمة ذات التلميحات من النوع NULL، أو مثيلاً لفئة معينة أو فئة فرعية من الفئة، أو مصفوفة، أو قابلة للاستدعاء كمعامل محدد. خلاف ذلك، يحدث خطأ وقت التشغيل.

يمكنك تحديد نوع بيانات لخاصية في فئة.

# إرجاع القيم

يمكن لدوال PHP أن ترجع قيمة واحدة فقط باستخدام الكلمة الرئيسية return:

```
function returnOne()
{
 return 42;
}
```

لإرجاع قيم متعددة، قم بإرجاع مصفوفة:

```
function returnTwo()
{
 return array("Fred", 35);
}
```

إذا لم يتم توفير قيمة إرجاع بواسطة دالة، تقوم الدالة بإرجاع NULL بدلاً من ذلك. يمكنك تعيين نوع بيانات الإرجاع بالتصريح عنه في تعريف الدالة. على سبيل المثال، سيرجع الكود التالي عددًا صحيحًا من 50 عند تنفيذه:

```
function someMath($var1, $var2): int
{
 return $var1 * $var2;
}


echo someMath(10, 5);
```

بشكل افتراضي، يتم نسخ القيم من الدالة. لإرجاع قيمة من خلال المرجع، قم بإرفاق اسم الدالة مسبقًا بكل من & عند التصريح عنها وعند تعيين القيمة المرتجعة إلى متغير:

```
$names = array("Fred", "Barney", "Wilma", "Betty");

function &findOne($n) {
 global $names;

 return $names[$n];
}

$person =& findOne(1); // Barney
$person = "Barnetta"; // changes $names[1]
```

في هذا الكود، ترجع الدالة findOne() اسمًا مستعارًا لـ $names[1] بدلاً من نسخة من قيمتها. نظرًا لأننا نسندها عن طريق المرجع، فإن $person هو اسم مستعار لـ $names[1] ، وتغير المهمة الثانية القيمة في $names[1].

تُستخدم هذه التقنية أحيانًا لإرجاع قيم سلسلة أو مصفوفة كبيرة بكفاءة من دالة. ومع ذلك، تنفذ PHP نسخًا عند الكتابة لقيم متغيرة، مما يعني أن إرجاع مرجع من دالة عادةً ما يكون غير ضروري. إرجاع مرجع إلى قيمة يكون أبطأ من إرجاع القيمة نفسها.

# دوال متغيرة

كما هو الحال مع المتغيرات المتغيرة حيث يشير التعبير إلى قيمة المتغير الذي يكون اسمه هو القيمة التي يحتفظ بها المتغير الظاهر (بنية $$)، يمكنك إضافة أقواس بعد متغير لاستدعاء الدالة التي يكون اسمها هو القيمة التي يحتفظ بها المتغير الظاهر - على سبيل المثال، ()variable$. ضع في اعتبارك هذا الموقف، حيث يتم استخدام متغير لتحديد أي من الدوال الثلاث التي يجب استدعاؤها:

```
switch ($which) {
 case 'first':
 first();
 break;

 case 'second':
 second();
 break;

 case 'third':
 third();
 break;
}
```

في هذه الحالة، يمكننا استخدام استدعاء دالة متغيرة لاستدعاء الدالة المناسبة. لإجراء استدعاء دالة متغيرة، قم بتضمين معلمات الدالة بين قوسين بعد المتغير. لإعادة كتابة المثال السابق:

```
$which(); // if $which is "first", the function first() is
called, etc...
```

في حالة عدم وجود دالة للمتغير، يحدث خطأ في وقت التشغيل عند تقييم الكود. لمنع حدوث ذلك، قبل استدعاء الدالة، يمكنك استخدام دالة function_exists() المضمنة لتحديد ما إذا كانت الدالة موجودة لقيمة المتغير:

```
$yesOrNo = function_exists(function_name);
```

كمثال:

```
if (function_exists($which)) {
 $which(); // if $which is "first", the function first() is called, etc...
}
```

لا يمكن استدعاء تراكيب اللغة مثل echo() و isset() من خلال الدوال المتغيرة:

```
$which = "echo";
$which("hello, world"); // does not work
```

# دوال مجهولة

تقوم بعض دوال PHP بجزء من عملها باستخدام دالة توفرها لهم. على سبيل المثال، تستخدم الدالة usort() دالة تقوم بإنشائها وتمريرها إليها كمعامل لتحديد ترتيب فرز العناصر في المصفوفة.

على الرغم من أنه يمكنك تحديد دالة لهذه الأغراض، كما هو موضح سابقًا، تميل هذه الدوال إلى أن تكون مترجمة ومؤقتة. لتعكس الطبيعة المؤقتة لرد الاتصال، قم بإنشاء واستخدام دالة مجهولة (تُعرف أيضًا باسم *closure*).

يمكنك إنشاء دالة مجهولة باستخدام صيغة تعريف الدالة العادية، ولكن يمكنك تعيينها إلى متغير أو تمريرها مباشرة.

يوضح المثال 3-6 مثالاً باستخدام usort().

مثال 3-6. وظائف مجهولة

```
$array = array("really long string here, boy", "this",
"middling length", "larger");

usort($array, function($a, $b) {
 return strlen($a) - strlen($b);
});
```

```
print_r($array);
```

يتم فرز المصفوفة حسب ()usort باستخدام الدالة المجهولة، بترتيب طول السلسلة.

يمكن للدوال المجهولة استخدام المتغيرات المحددة في نطاق التضمين الخاص بها باستخدام صيغة use. فمثلا:

```
$array = array("really long string here, boy", "this",
"middling length",
"larger");
$sortOption = 'random';

usort($array, function($a, $b) use ($sortOption)
{
 if ($sortOption == 'random') {
 // sort randomly by returning (-1, 0, 1) at random
 return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
});

print_r($array);
```

لاحظ أن دمج المتغيرات من نطاق التضمين ليس هو نفسه استخدام المتغيرات العامة - المتغيرات العامة تكون دائمًا في النطاق العام، بينما يسمح دمج المتغيرات للإغلاق باستخدام المتغيرات المحددة في النطاق المرفق. لاحظ أيضًا أن هذا ليس بالضرورة نفس النطاق الذي يتم فيه استدعاء الإغلاق. فمثلا:

```
$array = array("really long string here, boy", "this",
"middling length",
"larger");
$sortOption = "random";

function sortNonrandom($array)
{
 $sortOption = false;

 usort($array, function($a, $b) use ($sortOption)
 {
 if ($sortOption == "random") {
 // sort randomly by returning (-1, 0, 1) at random
 return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
 });

 print_r($array);
}
```

```
print_r(sortNonrandom($array));
```

في هذا المثال، يتم فرز $array بشكل طبيعي، وليس عشوائيًا - قيمة $sortOption داخل الإغلاق هي قيمة $sortOption في نطاق ()sortNonrandom، وليست قيمة $sortOption في النطاق العام.

# مالتالي

يمكن أن تكون الدوال التي يحددها المستخدم مربكة في الكتابة ومعقدة لتصحيح الأخطاء، لذا تأكد من اختبارها جيدًا ومحاولة قصرها على أداء مهمة واحدة لكل منها. في الفصل التالي سنلقي نظرة على السلاسل وكل ما تنطوي عليه، وهو موضوع آخر معقد ومربك. لا تثبط عزيمتك: تذكر أننا نبني أسسًا قوية لكتابة كود PHP جيد ومتين وموجز. بمجرد أن يكون لديك فهم قوي للمفاهيم الأساسية للدوال، والسلاسل، والمصفوفات، والكائنات، ستكون في طريقك لتصبح مطور PHP جيد.

# الفصل الرابع: السلاسل النصية

ستكون معظم البيانات التي تصادفها أثناء البرمجة عبارة عن سلاسل من الأحرف أو سلاسل. يمكن أن تحتوي السلاسل على أسماء الأشخاص وكلمات المرور والعناوين وأرقام بطاقات الائتمان وروابط الصور وسجل الشراء وغير ذلك. لهذا السبب، PHP لديها مجموعة واسعة من الدوال للعمل مع السلاسل.

يوضح هذا الفصل الطرق العديدة لإنشاء سلاسل في برامجك، بما في ذلك موضوع الاستيفاء الصعب أحيانًا (وضع قيمة متغير في سلسلة)، ثم يغطي دوال التغيير، والاقتباس، والتلاعب، والبحث في السلاسل. بنهاية هذا الفصل، ستكون خبيرًا في التعامل مع السلاسل.

# عن ثوابت السلسلة

هناك أربع طرق لكتابة سلسلة حرفية في كود PHP الخاص بك: استخدام علامات الاقتباس الفردية، وعلامات الاقتباس المزدوجة، وتنسيق المستند الموجود هنا (heredoc) المشتق من صدفة Unix، ووثيقة "ابن عمها" الآن (nowdoc). تختلف هذه الطرق فيما إذا كانت تتعرف على تسلسلات هروب خاصة تتيح لك ترميز أحرف أخرى أو استيفاء المتغيرات.

## توسيع المتغير Variable Interpolation

عندما تقوم بتعريف سلسلة حرفية باستخدام علامات اقتباس مزدوجة أو heredoc، فإن السلسلة تخضع لتوسيع المتغير "*variable interpolation*". التوسيع "Interpolation" هي عملية استبدال أسماء المتغيرات في السلسلة بقيمها المضمنة. هناك طريقتان لتوسيع المتغيرات في سلاسل.

أبسط الطريقتين هي وضع اسم المتغير في سلسلة نصية ذات علامة اقتباس مزدوجة أو heredoc:

```
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
Kilroy was here
```

الطريقة الأخرى هي إحاطة المتغير الذي يتم توسيعه بأقواس مجعدة. يضمن استخدام بناء الجملة هذا توسيع المتغير الصحيح. الاستخدام الكلاسيكي للأقواس المتعرجة هو إزالة الغموض عن اسم المتغير من أي نص محيط:

```
$n = 12;
echo "You are the {$n}th person";
You are the 12th person
```

بدون الأقواس المتعرجة، ستحاول PHP طباعة قيمة المتغير $nth.

على عكس بعض بيئات الصدفة، في PHP، لا تتم معالجة السلاسل بشكل متكرر للتوسيع. بدلاً من ذلك، تتم معالجة أي عمليات توسيع في سلسلة ذات علامات اقتباس مزدوجة أولاً ويتم استخدام النتيجة كقيمة للسلسلة:

```
$bar = 'this is not printed';
$foo = '$bar'; // single quotes
print("$foo");
$bar
```

## سلاسل ذات علامة اقتباس واحدة

السلاسل ذات علامات الاقتباس المفردة و nowdocs لا توسع المتغيرات. وبالتالي، لا يتم توسيع اسم المتغير في السلسلة التالية لأن السلسلة الحرفية التي تحدث فيها ذات علامات اقتباس مفردة:

```
$name = 'Fred';
$str = 'Hello, $name'; // single-quoted
echo $str;
Hello, $name
```

تسلسلات الهروب الوحيدة التي تعمل في سلاسل ذات علامات اقتباس مفردة هي '\، والتي تضع علامة اقتباس واحدة في سلسلة ذات علامات اقتباس واحدة، و \\، التي تضع شرطة مائلة للخلف في سلسلة ذات علامات اقتباس مفردة. يتم تفسير أي ظهور آخر للشرطة المائلة للخلف ببساطة على أنه شرطة مائلة للخلف:

```
$name = 'Tim O\'Reilly';// escaped single quote
echo $name;
$path = 'C:\\WINDOWS'; // escaped backslash
echo $path;
$nope = '\n'; // not an escape
echo $nope;
Tim O'Reilly
C:\WINDOWS
\n
```

## سلاسل مزدوجة الاقتباس

سلاسل ذات علامات اقتباس مزدوجة توسع المتغيرات وتوسع العديد من سلاسل هروب PHP. يسرد الجدول 4-1 تسلسلات الهروب التي تم التعرف عليها بواسطة PHP في سلاسل ذات علامات اقتباس مزدوجة.

الجدول 4-1. الهروب من التسلسلات في سلاسل ذات علامات اقتباس مزدوجة

| يمثل الحرف | سلاسل الهروب |
| --- | --- |
| علامة تنصيص مزدوجة | \" |
| سطر جديد | \n |
| Carriage return | \r |
| مسافة جدول | \t |
| Backslash | \\ |
| علامة الدولار | \$ |
| Left curly brace | \{ |
| Right curly brace | \} |
| Left square bracket | \[ |
| Right square bracket | \] |
| يتم تمثيل حروف ASCII بقيمة ثمانية | \0 through \777 |
| يتم تمثيل حرف ASCII بقيمة سداسية عشرية | \x0 through \xFF |

| سلاسل الهروب | يمثل الحرف |
| --- | --- |
| \u | ترميز UTF-8 |

إذا تم العثور على تسلسل هروب غير معروف (على سبيل المثال، شرطة مائلة للخلف متبوعًا بحرف ليس واحدًا من تلك الموجودة في الجدول 4-1) في سلسلة حرفية ذات علامات اقتباس مزدوجة، فسيتم تجاهلها (إذا كان لديك مستوى تحذير مجموعة E_NOTICE، يتم إنشاء تحذير لتسلسل هروب غير معروف):

```
$str = "What is \c this?";// unknown escape sequence
echo $str;
What is \c this?
```

### هنا المستندات Here Documents

يمكنك بسهولة وضع سلاسل متعددة الأسطر في البرنامج الخاص بك باستخدام heredoc، على النحو التالي:

```
$clerihew = <<< EndOfQuote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.

EndOfQuote;
echo $clerihew;
```

```
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
```

يخبر رمز المعرف >>> المحلل اللغوي لـ PHP أنك تكتب heredoc. يمكنك اختيار المعرّف (EndOfQuote في هذه الحالة)، ويمكنك وضعه بين علامتي اقتباس إذا كنت ترغب في ذلك (على سبيل المثال ، "EndOfQuote"). يبدأ السطر التالي النص الذي يقتبس من قبل heredoc، والذي يستمر حتى يصل إلى سطر يحتوي على المعرف فقط. لضمان عرض النص المقتبس في منطقة الإخراج تمامًا كما وضعته، شغّل وضع النص العادي عن طريق إضافة هذا الأمر في الجزء العلوي من ملف التعليمات البرمجية:

```
header('Content-Type: text/plain;');
```

بالتناوب، إذا كنت تتحكم في إعدادات الخادم الخاص بك، يمكنك تعيين default_mimetype على عادي في ملف php.ini:

```
default_mimetype = "text/plain"
```

ومع ذلك، لا يُنصح بهذا، لأنه يضع كل المخرجات من الخادم في وضع النص العادي، مما قد يؤثر على تخطيط معظم كود الويب الخاص بك.

إذا لم تقم بتعيين وضع النص العادي الخاص بك، فإن الوضع الافتراضي هو عادةً وضع HTML، والذي يعرض ببساطة الإخراج كله في سطر واحد.

عند استخدام heredoc لتعبير بسيط، يمكنك وضع فاصلة منقوطة بعد معرّف الإنهاء لإنهاء الجملة (كما هو موضح في المثال الأول). ومع ذلك، إذا كنت تستخدم heredoc في تعبير أكثر تعقيدًا، فستحتاج إلى متابعة التعبير في السطر التالي، كما هو موضح هنا:

```
printf(<<< Template
%s is %d years old.
Template
, "Fred", 35);
```

يتم الاحتفاظ بعلامات الاقتباس المفردة والمزدوجة في heredoc:

```
$dialogue = <<< NoMore
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
NoMore;
echo $dialogue;
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
```

كما هي المسافة البيضاء:

```
$ws = <<< Enough
 boo
 hoo
Enough;
// $ws = " boo\n hoo";
```

الجديد في PHP 7.3 هو المسافة البادئة لفاصل النهاية. يتيح ذلك تنسيقًا أكثر وضوحًا في حالة التعليمات البرمجية المضمنة، كما هو الحال في الدالة التالية:

```
function sayIt() {
 $ws = <<< "StufftoSay"
 The quick brown fox
 Jumps over the lazy dog.
 StufftoSay;
return $ws;
}

echo sayIt() ;
```

```
    The quick brown fox
    Jumps over the lazy dog.
```

يتم إزالة السطر الجديد قبل نهاية النهاية، لذلك فإن هاتين التخصيصتين متطابقتين:

```
$s = 'Foo';
// same as
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
```

إذا كنت تريد سطرًا جديدًا لإنهاء السلسلة المقتبسة، فستحتاج إلى إضافة واحدة بنفسك:

```
$s = <<< End
Foo

End;
```

# سلاسل الطباعة

هناك أربع طرق لإرسال الإخراج إلى المتصفح. تتيح لك بنية echo طباعة العديد من القيم دفعة واحدة، بينما تطبع ()print قيمة واحدة فقط. تقوم الدالة ()printf ببناء سلسلة منسقة عن طريق إدخال قيم في قالب. دالة ()print_r مفيدة في التصحيح؛ يقوم بطباعة محتويات المصفوفات والكائنات وأشياء أخرى في شكل مقروء إلى حد ما.

echo

لوضع سلسلة في HTML لصفحة تم إنشاؤها بواسطة PHP، استخدم echo. بينما تبدو - وفي الغالب تتصرف - مثل دالة، فإن echo هو بناء لغة. هذا يعني أنه يمكنك حذف الأقواس، وبالتالي فإن التعبيرات التالية متساوية:

```
echo "Printy";
echo("Printy"); // also valid
```

يمكنك تحديد عدة عناصر للطباعة عن طريق فصلها بفاصلات:

```
echo "First", "second", "third";
Firstsecondthird
```

يعد استخدام الأقواس عند محاولة تكرار قيم متعددة خطأً في التحليل:

```
// this is a parse error
echo("Hello", "world");
```

نظراً لأن echo ليست دالة حقيقية، فلا يمكنك استخدامه كجزء من تعبير أكبر:

```
// parse error
if (echo("test")) {
 echo("It worked!");
}
```

يمكنك بسهولة معالجة مثل هذه الأخطاء باستخدام دالة print() أو printf().

print()

ترسل الدالة print() قيمة واحدة (مدخلها) إلى المتصفح:

```
if (print("test\n")) {
 print("It worked!");
}
test
It worked!
```

printf()

تقوم الدالة ()printf بإخراج سلسلة مبنية عن طريق استبدال القيم في قالب (سلسلة التنسيق *format* " *string*"). مشتق من الدالة التي تحمل الاسم نفسه في مكتبة C القياسية. المدخل الأول لـ ()printf هي سلسلة التنسيق. المدخلات المتبقية هي القيم المطلوب استبدالها. يشير الحرف % A في سلسلة التنسيق إلى الاستبدال "substitution".

## معدِّلات التنسيق "FORMAT MODIFIERS"

تتكون كل علامة استبدال في القالب من علامة النسبة المئوية (٪)، وربما يتبعها معدِّلات من القائمة التالية، وتنتهي بمحدد نوع. (استخدم ٪٪ للحصول على حرف نسبة مئوية واحد في الإخراج.) يجب أن تظهر المعدلات بالترتيب الذي تم سردها به هنا:

1. محدد الحشو يشير إلى الحرف المراد استخدامه لتضمين النتائج لحجم السلسلة المناسب. حدد 0، أو مسافة، أو أي حرف مسبوق بعلامة اقتباس واحدة. الحشو بمسافات هو الافتراضي.

2. علامة. هذا له تأثير مختلف على السلاسل من تأثير على الأرقام. بالنسبة إلى السلاسل، يجبر الطرح (-) هنا على أن تكون السلسلة مضبوطة لليسار (الافتراضي هو ضبط لليمين). بالنسبة للأرقام، تفرض علامة الجمع (+) هنا طباعة الأرقام الموجبة بعلامة الجمع البادئة (على سبيل المثال، ستتم طباعة 35 كـ 35+).

3. الحد الأدنى لعدد الأحرف التي يجب أن يحتوي عليها هذا العنصر. إذا كانت النتيجة أقل من هذا العدد من الأحرف، فإن محدد الإشارة والحشو يتحكم في كيفية تغطية هذا الطول.

4. بالنسبة للأرقام العشرية، محدد دقيق يتكون من نقطة ورقم، هذا يحدد عدد الأرقام العشرية التي سيتم عرضها. بالنسبة للأنواع غير المضاعفة، يتم تجاهل هذا المحدد.

## محددات النوع

يخبر محدد النوع ()printf عن نوع البيانات التي يتم استبدالها. هذا يحدد تفسير المعدلات المدرجة سابقًا. هناك ثمانية أنواع، كما هو مدرج في الجدول 4-2.

الجدول 4-2. ()printf نوع محددات

| المحدد | المعنى |
|---|---|
| % | يعرض علامة النسبة المئوية. |
| b | المدخل هو عدد صحيح ويتم عرضه كرقم ثنائي. |
| c | المدخل هو عدد صحيح ويتم عرضه كحرف بهذه القيمة. |
| d | المدخل هو عدد صحيح ويتم عرضه كرقم عشري. |
| e | المدخل مزدوج "double" ويتم عرضه في تدوين علمي "scientific notation". |
| E | المدخل مزدوج ويتم عرضه في شكل علمي باستخدام الأحرف الكبيرة. |

| المحدد | المعنى |
| --- | --- |
| F | المدخل عبارة عن رقم عشري ويتم عرضه على هذا النحو بالتنسيق المحلي الحالي. |
| F | المدخل هو رقم عشري ويتم عرضه على هذا النحو "such" |
| g | المدخل عبارة عن مزدوج ويتم عرضه إما بالتدوين العلمي (كما هو الحال مع محدد النوع e%) أو كرقم عشري (كما هو الحال مع محدد النوع f%)، أيهما أقصر. |
| G | المدخل مزدوج ويتم عرضه إما بالتدوين العلمي (كما هو الحال مع محدد النوع E%) أو كرقم عشري (كما هو الحال مع محدد النوع f%)، أيهما أقصر. |
| o | المدخل هو عدد صحيح ويتم عرضه كرقم ثماني (الأساس 8). |
| s | المدخل عبارة عن سلسلة ويتم عرضه على هذا النحو. |
| u | المدخل عدد صحيح بدون إشارة ويتم عرضه كرقم عشري. |
| x | المدخل عدد صحيح ويتم عرضه كرقم سداسي عشري (الأساس -16)؛ يتم استخدام الأحرف الصغيرة. |
| X | المدخل عدد صحيح ويتم عرضه كرقم سداسي عشري (الأساس -16)؛ يتم استخدام الأحرف الكبيرة. |

تبدو دالة ()printf معقدة بشكل شنيع للأشخاص الذين ليسوا مبرمجين من النوع C، بمجرد أن تعتاد عليها، ستجدها أداة تنسيق قوية، وهنا بعض الأمثلة:

- ❖ رقم عشري لأقرب منزلتين عشريتين:

```
printf('%.2f', 27.452);
27.45
```

- ❖ إخراج عشري وسداسي عشري:

```
printf('The hex value of %d is %x', 214, 214);
The hex value of 214 is d6
```

- ❖ ترك عدد صحيح لثلاث منازل عشرية:

```
printf('Bond. James Bond. %03d.', 7);
Bond. James Bond. 007.
```

- ❖ تنسيق التاريخ:

```
printf('%02d/%02d/%04d', $month, $day, $year);
02/15/2005
```

- ❖ النسب المئوية:

```
printf('%.2f%% Complete', 2.1);
2.10% Complete
```

❖ حشو رقم فاصلة عائمة:

```
printf('You\'ve spent $%5.2f so far', 4.1);
You've spent $ 4.10 so far
```

تأخذ الدالة ()sprintf نفس المدخلات مثل ()printf ولكنها تُرجع السلسلة المضمنة بدلاً من طباعتها. يتيح لك هذا حفظ السلسلة في متغير لاستخدامها لاحقًا:

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);
// now we can interpolate $date wherever we need a date
```

### ()print_r و ()var_dump

تعرض الدالة ()print_r بذكاء ما يتم تمريره إليها، بدلاً من تحويل كل شيء إلى سلسلة، كما يفعل echo و ()print. يتم ببساطة طباعة السلاسل والأرقام. تظهر المصفوفات كقوائم بين قوسين من المفاتيح والقيم، مسبوقة بالمصفوفة:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' =>
'Wilma');
print_r($a);
Array
(
 [name] => Fred
 [age] => 35
 [wife] => Wilma)
```

يؤدي استخدام ()print_r في المصفوفة إلى تحريك المكرر الداخلي إلى موضع العنصر الأخير في المصفوفة. انظر الفصل 5 لمزيد من التكرارات والمصفوفات.

عندما تقوم ()print_r كائناً، ترى الكلمة Object، متبوعة بخصائص الكائن التي تمت تهيئتها معروضة كمصفوفة:

```
class P {
 var $name = 'nat';
 // ...
}

$p = new P;
print_r($p);
Object
(
 [name] => nat)
```

لا يتم عرض القيم المنطقية و NULL بشكل مفيد بواسطة ()print_r:

```
print_r(true); // prints "1";
1
print_r(false); // prints "";

print_r(null); // prints "";
```

لهذا السبب، يُفضل var_dump() على print_r() من أجل التصحيح. تعرض الدالة var_dump() أي قيمة PHP بتنسيق يمكن للبشر قراءته:

```
var_dump(true);
var_dump(false);
var_dump(null);
var_dump(array('name' => "Fred", 'age' => 35));
class P {
 var $name = 'Nat';
 // ...
}
$p = new P;
var_dump($p);
bool(true)
bool(false)
bool(null)
array(2) {
 ["name"]=>
 string(4) "Fred"
 ["age"]=>
 int(35)
}
object(p)(1) {
 ["name"]=>
 string(3) "Nat"
}
```

احذر من استخدام ()print_r أو ()var_dump على بنية متكررة مثل GLOBALS$ (الذي يحتوي على إدخال لـ GLOBALS يشير إلى نفسه). تتكرر الدالة ()print_r بلا حدود، بينما يتم قطع ()var_dump بعد زيارة نفس العنصر ثلاث مرات.

# الوصول إلى الأحرف الفردية

ترجع الدالة ()strlen عدد الأحرف في سلسلة:

```
$string = 'Hello, world';
$length = strlen($string); // $length is 12
```

يمكنك استخدام بناء جملة إزاحة السلسلة في سلسلة لمعالجة الأحرف الفردية:

```
$string = 'Hello';
for ($i=0; $i < strlen($string); $i++) {
 printf("The %dth character is %s\n", $i, $string{$i});
}
The 0th character is H
The 1th character is e
The 2th character is l
The 3th character is l
The 4th character is o
```

# سلاسل التنظيف

في كثير من الأحيان، يجب تنظيف السلاسل التي نحصل عليها من الملفات أو المستخدمين قبل أن نتمكن من استخدامها. هناك مشكلتان شائعتان في البيانات الأولية وهما وجود مسافة بيضاء غير ضرورية وحروف كبيرة غير صحيحة (الأحرف الكبيرة مقابل الأحرف الصغيرة).

## إزالة المسافة البيضاء

يمكنك إزالة المسافة البيضاء البادئة أو اللاحقة باستخدام الدوال التالية: ()trim و ()ltrim و ()rtrim:

```
$trimmed = trim(string [, charlist ]);
$trimmed = ltrim(string [, charlist ]);
$trimmed = rtrim(string [, charlist ]);
```

ترجع ()trim نسخة من السلسلة "*string*" مع إزالة مسافة بيضاء من البداية والنهاية. ()ltrim (l لليسار) يفعل الشيء نفسه، لكنه يزيل المسافة البيضاء فقط من بداية السلسلة. ()rtrim (r لليمين) يزيل المسافة البيضاء فقط من نهاية السلسلة. مدخل *charlist* الاختياري عبارة عن سلسلة تحدد جميع الأحرف المراد إزالتها. يتم إعطاء الأحرف الافتراضية للإزالة في الجدول 4-3.

الجدول 4-3. تمت إزالة الأحرف الافتراضية عن طريق: *trim(), ltrim(), and rtrim()*

| الحرف | قيمة ASCII | المعنى |
|---|---|---|
| " " | 0x20 | مسافة |
| "\t" | 0x09 | مسافة جدول |
| "\n" | 0x0A | سطر جديد |
| "\r" | 0x0D | Carriage return |
| "\0" | 0x00 | NUL-byte |
| "\x0B" | 0x0B | Vertical tab |

كمثال:

```
$title = " Programming PHP \n";
$str1 = ltrim($title); // $str1 is "Programming PHP \n"
$str2 = rtrim($title); // $str2 is " Programming PHP"
$str3 = trim($title); // $str3 is "Programming PHP"
```

بالنظر إلى سطر من البيانات المفصولة بمسافة جدول، استخدم مدخل charlist لإزالة المسافة البيضاء البادئة أو اللاحقة دون حذف مسافات الجدول:

```
$record = " Fred\tFlintstone\t35\tWilma\t \n";
$record = trim($record, " \r\n\0\x0B");
```

```
// $record is "Fred\tFlintstone\t35\tWilma"
```

## تغيير الحالة

تحتوي PHP على العديد من الدوال لتغيير حالة السلاسل: يعمل ()strtolower و ()strtoupper على سلاسل كاملة، ويعمل ()ucfirst فقط على الحرف الأول من السلسلة، ويعمل ()ucwords على الحرف الأول من كل كلمة في سلسلة. تأخذ كل دالة سلسلة لتعمل عليها كمدخل وتقوم بإرجاع نسخة من تلك السلسلة، تم تغييرها بشكل مناسب. فمثلا:

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
fred flintstone
FRED FLINTSTONE
Barney rubble
Barney Rubble
```

إذا كان لديك سلسلة أحرف مختلطة تريد تحويلها إلى "حالة أحرف العنوان"، حيث يكون الحرف الأول من كل كلمة بأحرف كبيرة وبقية الأحرف بحروف صغيرة (ولست متأكدًا من حالة السلسلة في البداية)، استخدم مزيجًا من ()strtolower و ()ucwords:

```
print(ucwords(strtolower($string1)));
Fred Flintstone
```

## الترميز والهروب "Encoding and Escaping"

نظرًا لأن برامج PHP تتفاعل غالبًا مع صفحات HTML وعناوين الويب (URLs) وقواعد البيانات، فهناك دوال لمساعدتك في العمل مع هذه الأنواع من البيانات. HTML وعناوين الويب وأوامر قاعدة البيانات كلها سلاسل، لكن كل منها يتطلب أحرفًا مختلفة ليتم هروبها بطرق مختلفة. على سبيل المثال، يجب كتابة مسافة في عنوان الويب بصيغة 20%، بينما علامة أصغر من (<) في مستندات الـ HTML يجب أن تكتب كالتالي: <. تحتوي PHP على عدد من الدوال المضمنة للتحويل من وإلى هذه الترميزات.

## HTML

الأحرف الخاصة في HTML يتم تمثيلها بواسطة كيانات "*entities*" مثل: < (<) and & (&).

هناك نوعان من دوال PHP التي تحول الأحرف الخاصة في سلسلة إلى كياناتها: واحدة لإزالة علامات HTML والأخرى لاستخراج الأوسمة الوصفية "meta tags" فقط.

الكيان اقتباس جميع الأحرف الخاصة

تعمل الدالة ()htmlentities على تغيير جميع الأحرف التي تحتوي على معادلات كيان HTML إلى تلك المكافئات (باستثناء حرف المسافة). يتضمن ذلك علامة أقل من (<)، وعلامة أكبر من (>)، وعلامة العطف (&)، والأحرف المحركة.

**فمثلا:**

```
$string = htmlentities("Einstürzende Neubauten");
echo $string;
```

```
Einstürzende Neubauten
```

نسخة هروب الكيان ”entity-escaped“، ü (يُرى من خلال عرض المصدر)، يتم عرضه بشكل صحيح كـ ü في صفحة الويب المقدمة. كما ترى، لم يتم تحويل المسافة إلى  .

تأخذ الدالة ()htmlentities ما يصل إلى ثلاث مدخلات:

```
$output = htmlentities(input, flags, encoding);
```

تحدد معلمة الترميز ”*encoding*“ مجموعة الأحرف، إذا تم تقديمها. الإعداد الافتراضي هو "UTF-8". تتحكم معلمة الإشارات ”*flags*“ في ما إذا كان سيتم تحويل علامات الاقتباس المفردة والمزدوجة إلى نماذج كيان لها. يحول ENT_COMPAT (الافتراضي) علامات الاقتباس المزدوجة فقط، بينما يحول ENT_QUOTES كلا نوعي الاقتباس، بينما يحول ENT_NOQUOTES أيًا منهما. لا يوجد خيار لتحويل علامات الاقتباس المفردة فقط. فمثلا:

```
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;


$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;


$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes flashed.&lt;p&gt;
```

```
$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

## اقتباس الكيانات فقط أحرف نصية بلغة HTML

تقوم الدالة ()htmlspecialchars بتحويل أصغر مجموعة ممكنة من الكيانات لإنشاء HTML صالح. يتم تحويل الكيانات التالية:

- ❖ العطف "Ampersands" (&) تغير لـ &
- ❖ علامات تنصيص مزدوجة "Double quotes" (") تغير لـ "
- ❖ علامات تنصيص مفردة (') تغير لـ ' (في حالة تشغيل ENT_QUOTES، كما هو موضح في ()htmlentities)
- ❖ علامات أقل من (<) تغير لـ <
- ❖ علامات أكبر من (>) تغير لـ >

إذا كان لديك تطبيق يعرض البيانات التي أدخلها المستخدم في نموذج، فأنت بحاجة إلى تشغيل تلك البيانات من خلال ()htmlspecialchars قبل عرضها أو حفظها. إذا لم تقم بذلك، وأدخل المستخدم سلسلة مثل "angle < 30" أو "sturm & drang"، فسيعتقد المتصفح أن الأحرف الخاصة هي HTML، مما يؤدي إلى صفحة مشوهة.

مثل ()htmlentities، يمكن أن يستغرق ()htmlspecialchars ما يصل إلى ثلاث مدخلات:

```
$output = htmlspecialchars(input, [flags, [encoding]]);
```

المدخلات الأعلام "flasgs" والترميز "*encoding*" لها نفس المعنى الذي تستخدمه مع ()htmlentities.

لا توجد دوال خاصة للتحويل مرة أخرى من الكيانات إلى النص الأصلي، لأن هذا نادرًا ما يكون مطلوبًا. ومع ذلك، هناك طريقة بسيطة نسبيًا للقيام بذلك. استخدم الدالة ()get_html_translation_table لجلب جدول الترجمة المستخدم بواسطة أي من هاتين الدالتين في نمط اقتباس محدد. على سبيل المثال، للحصول على جدول الترجمة الذي تستخدمه ()htmlentities، قم بما يلي:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

للحصول على جدول ()htmlspecialchars في وضع ENT_NOQUOTES، استخدم:

```
$table = get_html_translation_table(HTML_SPECIALCHARS,
ENT_NOQUOTES);
```

الحيلة الجيدة هي استخدام جدول الترجمة هذا، وقلبه باستخدام ()array_flip، وإدخاله إلى ()strtr لتطبيقه على سلسلة، وبالتالي القيام بعكس قيم ()htmlentities بفعالية:

```
$str = htmlentities("Einstürzende Neubauten"); // now
it is encoded

$table = get_html_translation_table(HTML_ENTITIES);
$revTrans = array_flip($table);

echo strtr($str, $revTrans); // back to normal
Einstürzende Neubauten
```

يمكنك بالطبع أيضًا جلب جدول الترجمة وإضافة أي ترجمات أخرى تريدها، ثم إجراء ()strtr. على سبيل المثال، إذا كنت تريد ()htmlentities لترميز كل مسافة إلى  ، كنت ستفعل:

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = ' ';
$encoded = strtr($original, $table);
```

## إزالة أوسمة HTML

تزيل الدالة ()strip_tags أوسمة HTML من السلسلة:

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';
$output = strip_tags($input);
// $output is 'Howdy, &quot;Cowboy&quot;'
```

قد تأخذ الدالة مدخل ثاني يحدد سلسلة من الأوسمة لتركها في السلسلة. يتم أيضًا الاحتفاظ بأشكال إغلاق الأوسمة المدرجة في المعلمة الثانية:

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';
$output = strip_tags($input, '<b>');
// $output is 'The <b>bold</b> tags will stay'
```

لا يتم تغيير الخاصيات الموجودة في الأوسمة المحفوظة بواسطة ()strip_tags. نظرًا لأن خاصيات مثل style و onmouseover يمكن أن تؤثر على مظهر صفحات الويب وسلوكها، فإن الاحتفاظ ببعض الأوسمة باستخدام ()strip_tags لن يزيل بالضرورة احتمال إساءة الاستخدام.

## استخراج الأوسمة الوصفية "meta"

ترجع الدالة ()get_meta_tags مصفوفة من الأوسمة الوصفية لصفحة HTML، محددة كاسم ملف محلي أو عنوان URL. يصبح اسم وسم meta (الكلمات الرئيسية، والمؤلف، والوصف، وما إلى ذلك) هو المفتاح في المصفوفة، ويصبح محتوى وسم meta هي القيمة المقابلة:

```
$metaTags = get_meta_tags('http://www.example.com/');
echo "Web page made by {$metaTags['author']}";
Web page made by John Doe
```

الشكل العام للدالة هو:

```
$array = get_meta_tags(filename [, use_include_path]);
```

مرر قيمة true لـ use_include_path للسماح لـ PHP بمحاولة فتح الملف باستخدام مسار التضمين القياسي.

## عناوين URL

يوفر PHP دوال للتحويل من وإلى ترميز URL، مما يسمح لك بإنشاء عناوين URL وفك تشفيرها. يوجد في الواقع نوعان من ترميز URL يختلفان في كيفية تعاملهما مع المسافات. الأول (المحدد بواسطة RFC 3986) يعامل المسافة على أنها مجرد حرف غير قانوني آخر في عنوان URL ويقوم بترميزها كـ 20%. الثانية (تنفيذ النظام application/x-www-form-urlencoded) ترميز مسافة كـ + وتستخدم في بناء سلاسل الاستعلام.

لاحظ أنك لا تريد استخدام هذه الدوال على عنوان URL كامل، مثل http://www.example.com/hello، حيث إنها ستتخطى النقطتين والشرط لإنتاج:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

قم بتشفير عناوين URL الجزئية فقط (البت بعد *http://www.example.com/*) وإضافة البروتوكول واسم المجال لاحقًا.

## ترميز RFC 3986 وفك ترميزه

لتشفير سلسلة وفقًا لاتفاقيات URL، استخدم ()rawurlencode:

```
$output = rawurlencode(input);
```

تأخذ هذه الدالة سلسلة وتقوم بإرجاع نسخة بأحرف URL غير قانونية مشفرة في اصطلاح dd%.

إذا كنت تقوم بإنشاء مراجع نص تشعبي ديناميكيًا للروابط في صفحة ما، فستحتاج إلى تحويلها باستخدام ()rawurlencode:

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/{$output}";
http://localhost/Programming%20PHP
```

تقوم الدالة ()rawurldecode بفك تشفير السلاسل المشفرة بعنوان URL:

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
Programming PHP
```

### الاستعلام عن سلسلة الترميز

تختلف دوال ()urlencode و ()urldecode عن نظيراتها الأولية فقط من حيث أنها تقوم بترميز المسافات كعلامات جمع (+) بدلاً من التسلسل 20%. هذا هو تنسيق بناء سلاسل الاستعلام وقيم ملفات تعريف الارتباط. يمكن أن تكون هذه الدوال مفيدة في توفير عناوين URL تشبه النموذج في HTML. تقوم PHP تلقائيًا بفك تشفير سلاسل الاستعلام وقيم ملفات تعريف الارتباط، لذلك لا تحتاج إلى استخدام هذه الدوال لمعالجة هذه القيم. تعتبر الدوال مفيدة في إنشاء سلاسل الاستعلام:

```
$baseUrl = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $baseUrl . urlencode($query);
echo $url;

http://www.google.com/q=PHP+sessions+-cookies
```

### SQL

تتطلب معظم أنظمة قواعد البيانات أن يتم تجاوز القيم الحرفية في استعلامات SQL. نظام ترميز SQL بسيط جدًا - يجب أن تُسبَق علامات الاقتباس المفردة وعلامات الاقتباس المزدوجة و NUL بايت والشرطات المائلة للخلف بشرطة مائلة للخلف. تضيف دالة ()addslashes هذه الخطوط المائلة، وتقوم الدالة ()stripslashes بإزالتها:

```
$string = <<< EOF
```

```
"It's never going to work," she cried,
as she hit the backslash (\) key.
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
\"It\'s never going to work,\" she cried,
as she hit the backslash (\\) key.
"It's never going to work," she cried,
as she hit the backslash (\) key.
```

## ترميز سلاسل C

تتخطى الدالة ()addcslashes الأحرف العشوائية بوضع الشرطات المائلة العكسية أمامها. باستثناء الأحرف الموجودة في الجدول 4-4، يتم ترميز الأحرف التي تحتوي على قيم ASCII أقل من 32 أو أعلى 126 بقيمها الثمانية (على سبيل المثال، "002"). تُستخدم الدالتان ()addcslashes و ()stripcslashes مع أنظمة قواعد البيانات غير القياسية التي لها أفكارها الخاصة حول الأحرف التي يجب تجاوزها.

الجدول 4-4. الهروب من حرف واحد يتعرف عليه *()addcslashes* و*()stripcslashes*

| الترميز | قيمة ASCII |
| --- | --- |
| \a | 7 |
| \b | 8 |

| الترميز | قيمة ASCII |
|---|---|
| \t | 9 |
| \n | 10 |
| \v | 11 |
| \f | 12 |
| \r | 13 |

قم باستدعاء ()addcslashes باستخدام مدخلين — السلسلة المطلوب تشفيرها والأحرف المراد تجاوزها:

```
$escaped = addcslashes(string, charset);
```

حدد نطاقًا من الأحرف للتخلي عنه باستخدام البناء "..":

```
echo addcslashes("hello\tworld\n", "\x00..\x1fz..\xff");
hello\tworld\n
```

احذر من تحديد '0' أو 'a' أو 'b' أو 'f' أو 'n' أو 'r' أو 't' أو 'v' في مجموعة الأحرف، حيث سيتم تحويلها إلى '0\' و'a\' وما إلى ذلك. يتم التعرف على عمليات الهروب هذه بواسطة C و PHP وقد تسبب ارتباكًا.

تأخذ ()stripcslashes سلسلة وتعيد نسخة مع توسيع عمليات الهروب:

```
$string = stripcslashes(escaped);
```

كمثال:

```
$string = stripcslashes('hello\tworld\n');
// $string is "hello\tworld\n"
```

# مقارنة السلاسل

PHP لها عاملين وست دوال لمقارنة السلاسل مع بعضها البعض.

## مقارنات دقيقة

يمكنك مقارنة سلسلتين للمساواة مع عوامل == و ===. تختلف هذه العوامل في كيفية تعاملها مع المعاملات غير الخيطية. يلقي عامل == معاملات السلسلة إلى أرقام، لذلك يبلغ عن تساوي 3 و "3". نظراً لقواعد تحويل السلاسل إلى أرقام، فإنه يُبلغ أيضًا عن تساوي 3 و "3b"، حيث يتم استخدام جزء من السلسلة يصل إلى حرف غير رقمي فقط في عملية الصب. عامل === لا يلقي، ويعيد خطأ إذا اختلفت أنواع بيانات المدخلات:

```
$o1 = 3;
$o2 = "3";

if ($o1 == $o2) {
 echo("== returns true<br>");
}
if ($o1 === $o2) {
 echo("=== returns true<br>");
}
== returns true
```

عوامل المقارنة (<, <=, >, >=) تعمل أيضا على السلاسل النصية:

```
$him = "Fred";
$her = "Wilma";

if ($him < $her) {
```

```
 print "{$him} comes before {$her} in the alphabet.\n";
}
Fred comes before Wilma in the alphabet
```

ومع ذلك، فإن عوامل المقارنة تعطي نتائج غير متوقعة عند مقارنة السلاسل والأرقام:

```
$string = "PHP Rocks";
$number = 5;

if ($string < $number) {
 echo("{$string} < {$number}");
}
PHP Rocks < 5
```

عندما تكون إحدى مدخلات عامل المقارنة عبارة عن رقم، يتم تحويل المدخل الأخر إلى رقم. هذا يعني أن "PHP Rocks" يتم تحويلها إلى رقم، مما يعطي 0 (لأن السلسلة لا تبدأ برقم). نظراً لأن الرقم 0 أقل من 5، تطبع "PHP Rocks < 5".

لمقارنة سلسلتين بشكل صريح كسلاسل، مع تحويل الأرقام إلى سلاسل إذا لزم الأمر، استخدم الدالة strcmp():

```
$relationship = strcmp(string_1, string_2);
```

تُرجع الدالة عددًا أقل من 0 إذا كانت *string_1* تفرز قبل *string_2*، أو أكبر من 0 إذا كانت *string_2* تفرز قبل *string_1*، أو 0 إذا كانت هي نفسها:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
1
```

الاختلاف في ()strcmp هو ()strcasecmp، والذي يحول السلاسل إلى أحرف صغيرة قبل مقارنتها. المدخلات وقيم الإرجاع الخاصة به هي نفسها الخاصة بـ ()strcmp:

```
$n = strcasecmp("Fred", "frED"); // $n is 0
```

هناك اختلاف آخر في مقارنة السلسلة وهو مقارنة الأحرف القليلة الأولى فقط من السلسلة. تأخذ الدالتان ()strncmp و ()strncasecmp مدخل إضافي، العدد الأولي للأحرف المراد استخدامها للمقارنات:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

الاختلاف الأخير في هذه الدالات هو مقارنة الترتيب الطبيعي *"natural-order"* مع ()strnatcmp و ()strnatcasecmp، اللذان يأخذان نفس المدخلات مثل ()strcmp ويعيدان نفس أنواع القيم. تحدد مقارنة الترتيب الطبيعي الأجزاء الرقمية من السلاسل التي تتم مقارنتها وتفرز أجزاء السلسلة بشكل منفصل عن الأجزاء الرقمية.

يوضح الجدول 4-5 السلاسل بالترتيب الطبيعي وترتيب ASCII.

الجدول 4-5. الترتيب الطبيعي مقابل ترتيب ASCII

| ترتيب ASCII | الترتيب الطبيعي |
|---|---|
| pic1.jpg | pic1.jpg |
| pic10.jpg | pic5.jpg |
| pic5.jpg | pic10.jpg |
| pic50.jpg | pic50.jpg |

## المساواة التقريبية

توفر PHP العديد من الدوال التي تتيح لك اختبار ما إذا كانت هناك سلسلتان متساويتان تقريبًا — soundex()، metaphone()، similar_text()، و levenshtein():

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep,
$cost_del ]);
```

ينتج عن كل من خوارزميات Soundex و Metaphone سلسلة تمثل تقريبًا كيفية نطق الكلمة باللغة الإنجليزية. لمعرفة ما إذا كانت هناك سلسلتان متساويتان تقريبًا مع هذه الخوارزميات، قارن طريقة

نطقهما. يمكنك مقارنة قيم Soundex فقط بقيم Soundex وقيم Metaphone فقط بقيم Metaphone. تعد خوارزمية Metaphone أكثر دقة بشكل عام، كما يوضح المثال التالي:

```
$known = "Fred";
$query = "Phred";

if (soundex($known) == soundex($query)) {
 print "soundex: {$known} sounds like {$query}<br>";
}
else {
 print "soundex: {$known} doesn't sound like {$query}<br>";
}

if (metaphone($known) == metaphone($query)) {
 print "metaphone: {$known} sounds like {$query}<br>";
}
else {
 print "metaphone: {$known} doesn't sound like {$query}<br>";
}
soundex: Fred doesn't sound like Phred
metaphone: Fred sounds like Phred
```

ترجع الدالة ()similar_text عدد الأحرف المشتركة بين مدخل السلسلة. المدخل الثالثة، إن وجدت، هي متغير يتم فيه تخزين القاسم المشترك كنسبة مئوية:

```
$string1 = "Rasmus Lerdorf";
```

```
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $percent);
printf("They have %d chars in common (%.2f%%).",
$common, $percent);
They have 13 chars in common (89.66%).
```

تحسب خوارزمية Levenshtein التشابه بين سلسلتين بناءً على عدد الأحرف التي يجب عليك إضافتها أو استبدالها أو إزالتها لجعلها متشابهة. على سبيل المثال، يكون لكل من "cat" و "cot" مسافة Levenshtein تبلغ 1، لأنك تحتاج إلى تغيير حرف واحد فقط ("a" إلى "o") لجعلهما متماثلين:

```
$similarity = levenshtein("cat", "cot"); // $similarity
is 1
```

عادةً ما يكون قياس التشابه هذا أسرع في الحساب من ذلك المستخدم بواسطة دالة ()similar_text. اختياريًا، يمكنك تمرير ثلاث قيم إلى الدالة ()levenshtein لوزن الإدخالات والحذف والاستبدال بشكل فردي — على سبيل المثال، لمقارنة كلمة مقابل تقلص "contraction".

يزن هذا المثال بشكل مفرط عمليات الإدخال عند مقارنة سلسلة مقابل تقلصها المحتمل، لأن التقلصات يجب ألا تدخل أحرفًا:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

# التلاعب والبحث في السلاسل

PHP لديها العديد من الدوال للعمل مع السلاسل. الدوال الأكثر استخدامًا للبحث عن السلاسل وتعديلها هي تلك التي تستخدم التعبيرات النمطية لوصف السلسلة المعنية. لا تستخدم الدوال الموضحة في هذا القسم التعبيرات العادية - فهي أسرع من التعبيرات العادية، ولكنها تعمل فقط عندما تبحث عن سلسلة ثابتة (على سبيل المثال، إذا كنت تبحث عن "01/11/12" بدلاً من ذلك من "أي أرقام مفصولة بشرطة مائلة").

## السلاسل Substrings

إذا كنت تعرف مكان البيانات التي تهتم بها في سلسلة أكبر، فيمكنك نسخها باستخدام الدالة ()substr:

```
$piece = substr(string, start [, length ]);
```

مدخل البداية "*start*" هو الموضع في السلسلة "*string*" الذي يبدأ عنده النسخ، حيث يشير الرقم 0 إلى بداية السلسلة. وسيط الطول "*length*" هو عدد الأحرف المراد نسخها (الافتراضي هو النسخ حتى نهاية السلسلة). فمثلا:

```
$name = "Fred Flintstone";
$fluff = substr($name, 6, 4); // $fluff is "lint"
$sound = substr($name, 11); // $sound is "tone"
```

لمعرفة عدد المرات التي تحدث فيها سلسلة أصغر داخل سلسلة أكبر، استخدم ()substr_count:

```
$number = substr_count(big_string, small_string);
```

كمثال:

```
$sketch = <<< EndOfSketch
```

```
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
The word spam occurs 14 times.
```

تسمح الدالة substr_replace() بأنواع عديدة من تعديلات السلسلة:

```
$string = substr_replace(original, new, start [, length ]);
```

تستبدل الدالة الجزء *original* المشار إليه بـ *start* (0 يعني بداية السلسلة) وقيم *length* بالسلسلة *new*. إذا لم يتم إعطاء مدخل رابع، فإن substr_replace() يزيل النص من *start* إلى نهاية السلسلة.

على سبيل المثال:

```
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye", 5, 7);
// $farewell is "good bye citizen"
```

استخدم *length* 0 للإدراج دون حذف:

```
$farewell = substr_replace($farewell, "kind ", 9, 0);
// $farewell is "good bye kind citizen"
```

استخدم استبدال "" للحذف دون إدخال:

```
$farewell = substr_replace($farewell, "", 8);
// $farewell is "good bye"
```

إليك كيفية الإدراج في بداية السلسلة:

```
$farewell = substr_replace($farewell, "now it's time to
say ", 0, 0);
// $farewell is "now it's time to say good bye"'
```

تشير القيمة السالبة للبداية إلى عدد الأحرف من نهاية السلسلة التي تبدأ منها عملية الاستبدال:

```
$farewell = substr_replace($farewell, "riddance", -3);
// $farewell is "now it's time to say good riddance"
```

يشير *length* السالب إلى عدد الأحرف من نهاية السلسلة التي يجب عندها إيقاف الحذف:

```
$farewell = substr_replace($farewell, "", -8, -5);
// $farewell is "now it's time to say good
```

## دوال السلاسل المتنوعة

تأخذ الدالة ()strrev سلسلة وتعيد نسخة معكوسة منها:

```
$string = strrev(string);
```

كمثال:

```
echo strrev("There is no cabal");
labac on si erehT
```

تأخذ الدالة ()str_repeat سلسلة "string" وعدداً "count" وتعيد سلسلة جديدة تتكون من عدد "*count*"مرات تكرار سلسلة "*string*" المدخل:

```
$repeated = str_repeat(string, count);
```

على سبيل المثال، لبناء قاعدة أفقية مموجة:

```
echo str_repeat('_.-.', 40);
```

تقوم دالة ()str_pad بربط سلسلة مع أخرى. اختياريًا، يمكنك تحديد السلسلة التي سيتم ربطها، وما إذا كنت ستضعها على اليسار أو اليمين أو كليهما:

```
$padded = str_pad(to_pad, length [, with [, pad_type
]]);
```

الافتراضي هو ربط على اليمين بمسافات:

```
$string = str_pad('Fred Flintstone', 30);
echo "{$string}:35:Wilma";
Fred Flintstone :35:Wilma
```

المدخل الثالث الاختياري هي السلسلة المطلوب ربطها:

```
$string = str_pad('Fred Flintstone', 30, '. ');
echo "{$string}35";
Fred Flintstone. . . . . . . .35
```

يمكن أن يكون المدخل الرابع الاختياري STR_PAD_RIGHT (الافتراضي) أو STR_PAD_LEFT أو STR_PAD_BOTH (للتوسط). فمثلا:

```
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
[ Fred Flintstone]
[ Fred Flintstone ]
```

## تحليل سلسلة Decomposing a String

توفر PHP العديد من الدوال للسماح لك بتقسيم سلسلة إلى مكونات أصغر. بترتيب متزايد من التعقيد، هي: explode() و strtok() و sscanf().

## EXPLODING AND IMPLODING

غالبًا ما تصل البيانات كسلاسل، والتي يجب تقسيمها إلى مجموعة من القيم. على سبيل المثال، قد ترغب في تقسيم الحقول المفصولة بفواصل من سلسلة مثل "Fred,25,Wilma." في هذه الحالات، استخدم دالة explode():

```
$array = explode(separator, string [, limit]);
```

المدخل الأول، الفاصل "*separator*"، عبارة عن سلسلة تحتوي على فاصل الحقل. المدخل الثاني، سلسلة نصية "*string*"، هي السلسلة المراد تقسيمها. المدخل الثالث الاختياري، الحد "*limit*"، هي الحد الأقصى لعدد القيم التي يتم إرجاعها في المصفوفة. إذا تم الوصول إلى الحد، فإن العنصر الأخير في المصفوفة يحتوي على باقي السلسلة:

```
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
// $fields is array('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields is array('Fred', '25,Wilma')
```

تعمل دالة implode() على العكس تمامًا من explode() - فهي تنشئ سلسلة كبيرة من مجموعة من السلاسل الأصغر:

```
$string = implode(separator, array);
```

المدخل الأول، الفاصل "*separator*"، هو السلسلة التي يجب وضعها بين عناصر المدخل الثاني، المصفوفة "*array*". لإعادة بناء سلسلة القيمة البسيطة المفصولة بفاصلة، قل ببساطة:

```
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields); // $string is 'Fred,25,Wilma'
```

دالة ()join هي اسم مستعار لـ ()implode.

## TOKENIZING

تتيح لك دالة ()strtok التكرار خلال سلسلة، والحصول على قطعة جديدة (رمز مميز "token") في كل مرة. في المرة الأولى التي تستدعيها، تحتاج إلى تمرير مدخلين: السلسلة المراد تكرارها وفاصل الرمز المميز. فمثلا:

```
$firstChunk = strtok(string, separator);
```

لاسترداد بقية الرموز، قم باستدعاء ()strtok بشكل متكرر باستخدام الفاصل فقط:

```
$nextChunk = strtok(separator);
```

على سبيل المثال، ضع في اعتبارك هذا الاستدعاء:

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");
```

```
while ($token !== false) {
 echo("{$token}<br />");
 $token = strtok(",");
}
Fred
Flintstone
35
Wilma
```

ترجع الدالة ()false strtok عندما لا يكون هناك المزيد من الرموز التي يتم إرجاعها.

قم باستدعاء ()strtok مع مدخلين لإعادة تهيئة المكرر. يؤدي هذا إلى إعادة تشغيل الرمز المميز من بداية السلسلة.

## SSCANF()

تحلل الدالة ()sscanf سلسلة وفقًا لقالب مثل ()printf:

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

إذا تم استخدامه بدون المتغيرات الاختيارية، فإن ()sscanf ترجع مصفوفة من الحقول:

```
$string = "Fred\tFlintstone (35)";
$a = sscanf($string, "%s\t%s (%d)");
print_r($a);
```

```
Array
(
 [0] => Fred
 [1] => Flintstone
 [2] => 35)
```

قم بتمرير المراجع إلى المتغيرات لتخزين الحقول في تلك المتغيرات. يتم إرجاع عدد الحقول المخصصة:

```
$string = "Fred\tFlintstone (35)";
$n = sscanf($string, "%s\t%s (%d)", $first, $last,
$age);
echo "Matched {$n} fields: {$first} {$last} is {$age}
years old";
Matched 3 fields: Fred Flintstone is 35 years old
```

## دوال البحث في السلاسل

تجد العديد من الدالات سلسلة أو حرفًا داخل سلسلة أكبر. يأتون في ثلاث عائلات: ()strpos و ()strrpos، والتي ترجع المكان؛ ()strstr، ()strchr، والأصدقاء، الذين يعيدون السلسلة التي يعثرون عليها، و ()strspn و()strcspn، اللذان يعيدان مقدار ما يتطابق مع القناع في بداية السلسلة.

في جميع الحالات، إذا حددت رقمًا على أنه "سلسلة" للبحث عنه، فإن PHP تتعامل مع هذا الرقم باعتباره القيمة الترتيبية للحرف المراد البحث عنه. وبالتالي، فإن استدعاءات الدوال هذه متطابقة لأن 44 هي قيمة ASCII للفاصلة:

```
$pos = strpos($large, ","); // find first comma
$pos = strpos($large, 44); // also find first comma
```

جميع دوال البحث عن السلاسل ترجع false إذا لم تتمكن من العثور على السلسلة الفرعية التي حددتها. إذا كانت السلسلة الفرعية تحدث في بداية السلسلة، فإن الدالات ترجع 0. نظراً لأن false يلقي بالرقم 0، فقم دائمًا بمقارنة قيمة الإرجاع بـ === عند اختبار الفشل:

```
if ($pos === false) {
 // wasn't found
}
else {
 // was found, $pos is offset into string
}
```

## عمليات البحث بإرجاع الأماكن

تبحث الدالة ()strpos عن التواجد الأول لسلسلة نصية صغيرة في سلسلة أكبر:

```
$position = strpos(large_string, small_string);
```

إذا لم يتم العثور على السلسلة الصغيرة، فإن ()strpos ترجع false.

تبحث الدالة ()strrpos عن آخر تواجد لحرف في سلسلة. يأخذ نفس المدخلات ويرجع نفس نوع القيمة مثل ()strpos.

على سبيل المثال:

```
$record = "Fred,Flintstone,35,Wilma";
```

```
$pos = strrpos($record, ","); // find last comma
echo("The last comma in the record is at position {$pos}");
The last comma in the record is at position 18
```

## عمليات البحث التي عادت بقية السلسلة

تبحث الدالة ()strstr عن أول ظهور لسلسلة صغيرة في سلسلة أكبر وترجع من تلك السلسلة الصغيرة.
على سبيل المثال:

```
$record = "Fred,Flintstone,35,Wilma";
$rest = strstr($record, ","); // $rest is ",Flintstone,35,Wilma"
```

الاختلافات في ()strstr هي:

- *stristr*()

  غير حساس لحالة الأحرف ()strstr

- *strchr*()

  هو اسم مستعار لـ ()strstr

- *strrchr*()

  البحث عن آخر تكرار لحرف في سلسلة

كما هو الحال مع ()strrpos، يبحث ()strrchr للخلف في السلسلة، ولكن عن حرف واحد فقط، وليس عن سلسلة كاملة.

## عمليات البحث باستخدام الأقنعة

إذا كنت تعتقد أن ()strrchr مقصور على فئة معينة، فأنت لم تر شيئًا بعد. تخبرك الدالتان ()strspn و ()strcspn بعدد الأحرف في بداية سلسلة تتكون من أحرف معينة:

```
$length = strspn(string, charset);
```

على سبيل المثال، تختبر هذه الدالة ما إذا كانت السلسلة تحتوي على رقم ثماني:

```
function isOctal($str)
{
 return strspn($str, '01234567') == strlen($str);
}
```

يرمز c في ()strcspn إلى التكملة *"complement"* - فهو يخبرك كم من بداية السلسلة لا يتألف من الأحرف في مجموعة الأحرف. استخدمه عندما يكون عدد الأحرف الشيقة أكبر من عدد الأحرف غير المثيرة للاهتمام. على سبيل المثال، تختبر هذه الدالة ما إذا كانت السلسلة تحتوي على أي NUL بايت أو علامات جدولة أو حرف إرجاع:

```
function hasBadChars($str)
{
 return strcspn($str, "\n\t\0") != strlen($str);
}
```

## فك عناوين المواقع DECOMPOSING URLS

ترجع الدالة ()parse_url مجموعة من مكونات عنوان URL:

```
$array = parse_url(url);
```

كمثال:

```
$bits  =  parse_url("http://me:secret@example.com/cgi-
bin/board?user=fred");

print_r($bits);


Array

(

 [scheme] => http

 [host] => example.com

 [user] => me

 [pass] => secret

 [path] => /cgi-bin/board

 [query] => user=fred)
```

المفاتيح المحتملة للتجزئة هي المخطط وhost و port و user وpass وpath وquery وfragment.

# التعبيرات العادية Regular Expressions

إذا كنت بحاجة إلى دوال بحث أكثر تعقيدًا مما توفره الطرق السابقة، فيمكنك استخدام تعبير عادي - سلسلة تمثل نمطًا "*pattern*". تقارن دوال التعبير العادي هذا النمط بسلسلة أخرى ومعرفة ما إذا كان أي من السلسلة يتطابق مع النمط. تخبرك بعض الدوال بما إذا كان هناك تطابق، بينما يقوم البعض الآخر بإجراء تغييرات على السلسلة.

هناك ثلاثة استخدامات للتعبيرات النمطية: المطابقة، والتي يمكن استخدامها أيضًا لاستخراج المعلومات من سلسلة؛ استبدال نص جديد بنص مطابق؛ وتقسيم السلسلة إلى مجموعة من القطع الأصغر. PHP لها دوال للجميع. على سبيل المثال، يطابق preg_match() تعبيرًا عاديًا.

لطالما اعتبرت Perl معيارًا للتعبيرات النمطية القوية. تستخدم PHP مكتبة C تسمى *pcre* لتقديم دعم شبه كامل لترسانة Perl من ميزات التعبير العادي. تعمل تعبيرات Perl العادية على بيانات ثنائية عشوائية، لذا يمكنك المطابقة بأمان مع الأنماط أو السلاسل التي تحتوي على (\x00) NUL-byte.

## أساسيات

معظم الأحرف في التعبير العادي هي أحرف حرفية، مما يعني أنها تطابق نفسها فقط. على سبيل المثال، إذا كنت تبحث عن التعبير العادي "/cow/" في السلسلة "Dave was a cowhand"، فستحصل على تطابق لأن كلمة "cow" توجد في تلك السلسلة.

بعض الأحرف لها معاني خاصة في التعبيرات العادية. على سبيل المثال، تشير علامة الإقحام "caret" (^) في بداية التعبير النمطي إلى أنه يجب أن يتطابق مع بداية السلسلة (أو بشكل أكثر دقة، يربط "*anchors*" التعبير العادي ببداية السلسلة):

```
preg_match("/^cow/", "Dave was a cowhand"); // returns false
preg_match("/^cow/", "cowabunga!"); // returns true
```

وبالمثل، فإن علامة الدولار ($) في نهاية التعبير العادي تعني أنه يجب أن تتطابق مع نهاية السلسلة (على سبيل المثال، تثبت التعبير العادي في نهاية السلسلة):

```
preg_match("/cow$/", "Dave was a cowhand"); // returns false
preg_match("/cow$/", "Don't have a cow"); // returns true
```

النقطة (.) في التعبير العادي تطابق أي حرف مفرد:

```
preg_match("/c.t/", "cat"); // returns true
preg_match("/c.t/", "cut"); // returns true
preg_match("/c.t/", "c t"); // returns true
preg_match("/c.t/", "bat"); // returns false
preg_match("/c.t/", "ct"); // returns false
```

إذا كنت تريد مطابقة أحد هذه الأحرف الخاصة (تسمى الحرف الأولي "*metacharacter*")، فعليك الهروب منه بشرطة مائلة للخلف:

```
preg_match("/\$5.00/", "Your bill is $5.00 exactly");
// returns true
preg_match("/$5.00/", "Your bill is $5.00 exactly"); //
returns false
```

تكون التعبيرات العادية حساسة لحالة الأحرف بشكل افتراضي، لذلك لا يتطابق التعبير العادي "/cow/" مع السلسلة "COW". إذا كنت تريد إجراء مطابقة غير حساسة لحالة الأحرف، فإنك تحدد علامة للإشارة إلى ذلك (كما سترى لاحقًا في هذا الفصل).

حتى الآن، لم نقم بأي شيء لم يكن بوسعنا فعله مع دوال السلسلة التي رأيناها بالفعل، مثل ()strstr. تأتي القوة الحقيقية للتعبيرات العادية من قدرتها على تحديد أنماط مجردة يمكن أن تتطابق مع العديد من تسلسلات الأحرف المختلفة. يمكنك تحديد ثلاثة أنواع أساسية من الأنماط المجردة في تعبير عادي:

- ❖ مجموعة من الأحرف المقبولة التي يمكن أن تظهر في السلسلة (على سبيل المثال، الأحرف الأبجدية والأحرف الرقمية وعلامات الترقيم المحددة)
- ❖ مجموعة من البدائل للسلسلة (على سبيل المثال، "com" أو "edu" أو "net" أو "org")
- ❖ تسلسل متكرر في السلسلة (على سبيل المثال، حرف واحد على الأقل وليس أكثر من خمسة أحرف رقمية)

يمكن دمج هذه الأنواع الثلاثة من الأنماط بطرق لا حصر لها لإنشاء تعبيرات عادية تطابق أشياء مثل أرقام الهواتف وعناوين URL الصالحة.

## فئات الأحرف "Character Classes"

لتحديد مجموعة من الأحرف المقبولة في النمط الخاص بك، يمكنك إما إنشاء فئة حرف بنفسك أو استخدام فئة محددة مسبقًا. يمكنك بناء فئة الحرف الخاصة بك عن طريق تضمين الأحرف المقبولة بين قوسين مربعين:

```
preg_match("/c[aeiou]t/", "I cut my hand"); // returns true
preg_match("/c[aeiou]t/", "This crusty cat"); // returns true
preg_match("/c[aeiou]t/", "What cart?"); // returns false
preg_match("/c[aeiou]t/", "14ct gold"); // returns false
```

يبحث محرك التعبير العادي عن "c"، ثم يتحقق من أن الحرف التالي هو حرف "a" أو "e" أو "i" أو "o" أو "u". إذا لم يكن حرف متحرك، تفشل المطابقة ويرجع المحرك للبحث عن حرف "c" آخر. إذا تم العثور على حرف متحرك، فإن المحرك يتحقق من أن الحرف التالي هو "t". إذا كان الأمر كذلك، يكون المحرك في نهاية المباراة ويعود صحيحًا. إذا لم يكن الحرف التالي "t"، يعود المحرك للبحث عن "c" آخر.

يمكنك رفض فئة حرف باستخدام علامة الإقحام (^) في البداية:

```
preg_match("/c[^aeiou]t/", "I cut my hand"); // returns false
preg_match("/c[^aeiou]t/", "Reboot chthon"); // returns true
preg_match("/c[^aeiou]t/", "14ct gold"); // returns false
```

في هذه الحالة، يبحث محرك التعبير العادي عن "c" متبوعًا بحرف ليس حرف متحرك، متبوعًا بحرف "t".

يمكنك تحديد نطاق من الأحرف بواصلة "hyphen" (-). يعمل هذا على تبسيط فئات الأحرف مثل "جميع الأحرف "all letters"" و "جميع الأرقام "all digits"":

```
preg_match("/[0-9]%/", "we are 25% complete"); // returns true
preg_match("/[0123456789]%/", "we are 25% complete"); // returns true
preg_match("/[a-z]t/", "11th"); // returns false
preg_match("/[a-z]t/", "cat"); // returns true
preg_match("/[a-z]t/", "PIT"); // returns false
preg_match("/[a-zA-Z]!/", "11!"); // returns false
preg_match("/[a-zA-Z]!/", "stop!"); // returns true
```

عند تحديد فئة حرف، تفقد بعض الأحرف الخاصة معناها، بينما يأخذ البعض الآخر معاني جديدة. على وجه الخصوص، تفقد anchor $ والنقطة معناها في فئة الحرف، بينما لم يعد الحرف ^ رابطًا ولكنه يلغي فئة الحرف إذا كان هو الحرف الأول بعد القوس المفتوح. على سبيل المثال، يتطابق [[\^] مع أي حرف قوس غير مغلق، بينما يتطابق [^.$] مع أي علامة دولار أو نقطة أو علامة إقحام.

تحدد مكتبات التعبير العادي المتنوعة اختصارات لفئات الأحرف، بما في ذلك الأرقام والأحرف الأبجدية والمسافات البيضاء.

## البدائل "Alternatives"

يمكنك استخدام حرف التوجيه الرأسي (|) لتحديد البدائل في التعبير العادي:

```
preg_match("/cat|dog/", "the cat rubbed my legs"); // returns true
preg_match("/cat|dog/", "the dog rubbed my legs"); // returns true
preg_match("/cat|dog/", "the rabbit rubbed my legs"); // returns false
```

يمكن أن تكون أسبقية التناوب مفاجأة: "/^cat|dog$/" مختارة من "^cat" و "dog$"، بمعنى أنه يتطابق مع سطر يبدأ بـ "cat" أو تنتهي بـ "dog". إذا كنت تريد سطرًا يحتوي فقط على "cat" أو "dog" ، تحتاج لاستخدام التعبير العادي "/^(cat|dog)$/".

يمكنك الجمع بين فئات الأحرف والتبديل، على سبيل المثال، للتحقق من السلاسل التي لا تبدأ بحرف كبير:

```
preg_match("/^([a-z]|[0-9])/", "The quick brown fox"); // returns false
preg_match("/^([a-z]|[0-9])/", "jumped over"); // returns true
preg_match("/^([a-z]|[0-9])/", "10 lazy dogs"); // returns true
```

## تكرار المتواليات "Repeating Sequences"

لتحديد نمط متكرر، يمكنك استخدام مُحدِّد كمي "*quantifier*" . يتتبع المُحدِّد الكمي النمط الذي يتكرر ويحدد عدد المرات لتكرار هذا النمط. يوضح الجدول 4-6 المحددات الكمية التي تدعمها تعبيرات PHP العادية.

الجدول 4-6. محددات التعبير العادي

| المعنى | محدد الكم |
|---|---|
| 1 أو 0 | ? |
| 0 فأكثر | * |
| واحد وأكثر 1 | + |
| عدد مرات n بالضبط | { *n* } |
| على الأقل n، وليس أكثر من m مرة | { *n* , *m* } |
| على الأقل n مرات | { *n* ,} |

لتكرار حرف واحد، ضع المُحدد الكمي بعد الحرف:

```
preg_match("/ca+t/", "caaaaaaat"); // returns true
preg_match("/ca+t/", "ct"); // returns false
preg_match("/ca?t/", "caaaaaaat"); // returns false
preg_match("/ca*t/", "ct"); // returns true
```

باستخدام المحددات الكمية وفئات الأحرف، يمكننا فعلاً فعل شيء مفيد، مثل مطابقة أرقام الهواتف الأمريكية الصالحة:

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/",    "303-555-1212"); // returns true
```

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/",  "64-9-555-1234"); // returns false
```

## الأنماط الفرعية Subpatterns

يمكنك استخدام الأقواس لتجميع بتات التعبير العادي معًا لتتم معاملتها كوحدة واحدة تسمى النمط الفرعي *"subpattern"*:

```
preg_match("/a (very )+big dog/", "it was a very very big dog"); // returns true
preg_match("/^(cat|dog)$/", "cat"); // returns true
preg_match("/^(cat|dog)$/", "dog"); // returns true
```

تتسبب الأقواس أيضًا في التقاط السلسلة الفرعية التي تطابق النمط الفرعي. إذا قمت بتمرير مصفوفة كمتحول ثالث لدالة مطابقة، فسيتم ملء المصفوفة بأي سلاسل فرعية ملتقطة:

```
preg_match("/([0-9]+)/", "You have 42 magic beans", $captured);
// returns true and populates $captured
```

يتم تعيين العنصر الصفري للمصفوفة على السلسلة النصية المطابقة بالكامل. العنصر الأول هو السلسلة الفرعية التي تطابق النمط الفرعي الأول (إذا كان هناك واحد)، والعنصر الثاني هو السلسلة الفرعية التي تطابق النمط الفرعي الثاني، وهكذا.

## المحددات "Delimiters"

تحاكي التعبيرات العادية من نمط Perl صيغة Perl للنماذج، مما يعني أن كل نمط يجب أن يكون محاطًا بزوج من المحددات. تقليديا، يتم استخدام حرف الشرطة المائلة للأمام (/)؛ على سبيل المثال، */pattern/*. ومع ذلك، يمكن استخدام أي حرف غير أبجدي رقمي بخلاف حرف الخط المائل العكسي (\) لتحديد تنسيق نمط Perl. هذا مفيد لمطابقة السلاسل التي تحتوي على خطوط مائلة، مثل أسماء الملفات. على سبيل المثال، ما يلي متكافئ:

```
preg_match("/\/usr\/local\//", "/usr/local/bin/perl");
// returns true
preg_match("#/usr/local/#", "/usr/local/bin/perl"); //
returns true
```

يمكن استخدام الأقواس (()) والأقواس المتعرجة ({}) والأقواس المربعة ([]) وأقواس الزاوية (<>) كمحددات نمط:

```
preg_match("{/usr/local/}", "/usr/local/bin/perl"); //
returns true
```

يناقش قسم "الخيارات اللاحقة" المعدِّلات ذات الحرف الواحد التي يمكنك وضعها بعد محدد الإغلاق لتعديل سلوك محرك التعبير العادي. يعد x مفيدًا جدًا، مما يجعل شريط محرك التعبير العادي مسافة بيضاء و # التعليقات المميزة من التعبير العادي قبل المطابقة. هذان النمطان متماثلان، لكن أحدهما أسهل في القراءة:

```
'/([[:alpha:]]+)\s+\1/'
'/( # start capture
[[:alpha:]]+ # a word
\s+ # whitespace
\1 # the same word again
```

```
 ) # end capture
/x'
```

## سلوك المطابقة "Match Behavior"

النقطة (.) تطابق أي حرف باستثناء سطر جديد (\n). تتطابق علامة الدولار ($) في نهاية السلسلة، أو إذا انتهت السلسلة بسطر جديد، فقبل هذا السطر الجديد مباشرةً:

```
preg_match("/is (.*)$/", "the key is in my pants", $captured);
// $captured[1] is 'in my pants'
```

## فئات الأحرف "Character Classes"

كما هو موضح في الجدول 4-7، تحدد التعبيرات العادية المتوافقة مع Perl عددًا من مجموعات الأحرف المسماة التي يمكنك استخدامها في فئات الأحرف. التوسعات في الجدول 4-7 مخصصة للغة الإنجليزية. تختلف الأحرف الفعلية من لغة إلى أخرى.

يمكن استخدام فئة [: *something* :] بدلاً من حرف في فئة الحرف. على سبيل المثال، للعثور على أي حرف يمثل رقمًا أو حرفًا كبيرًا أو علامة "at" (@)، استخدم التعبير العادي التالي:

```
[@[:digit:][:upper:]]
```

ومع ذلك، لا يمكنك استخدام فئة الأحرف كنقطة نهاية النطاق:

```
preg_match("/[A-[:lower:]]/", "string");// invalid regular expression
```

بعض المناطق تعتبر تسلسلات أحرف معينة كما لو كانت حرفًا واحدًا – وتسمى هذه التسلسلات المتسلسلة "*collating sequences*". لمطابقة أحد هذه التسلسلات متعددة الأحرف في فئة الأحرف، قم بإرفاقها بـ [. و.]. على سبيل المثال، إذا كانت لغتك تحتوي على تسلسل الحروب ch، فيمكنك مطابقة s أو t أو ch بفئة الأحرف هذه:

```
[st[.ch.]]
```

الامتداد النهائي لفئات الأحرف هو فئة التكافؤ "*equivalence class*"، والتي تحددها من خلال تضمين الحرف داخل [= و =]. تطابق فئات التكافؤ الأحرف التي لها نفس ترتيب التجميع، كما هو محدد في الإعدادات المحلية الحالية. على سبيل المثال، قد تحدد الإعدادات المحلية a و á و ä على أنها لها نفس أسبقية الفرز. لمطابقة أي واحد منهم، فإن فئة التكافؤ هي [= a =].

الجدول 4-7. فئات الأحرف

| الفئة | الوصف | توسيع |
|---|---|---|
| [:alnum:] | أحرف أبجدية ورقمية | [0-9a-zA-Z] |
| [:alpha:] | الأحرف الأبجدية | [a-zA-Z] |
| [:ascii:] | 7-bit ASCII | [\x01-\x7F] |
| [:blank:] | Horizontal whitespace (space, tab) | [ \t] |

<table>
<tr><th>توسيع</th><th>الوصف</th><th>الفئة</th></tr>
<tr><td>[\x01-\x1F]</td><td>حروف التحكم</td><td>[:cntrl:]</td></tr>
<tr><td>[0-9]</td><td>أرقام</td><td>[:digit:]</td></tr>
<tr><td>[^\x01-\x20]</td><td>الأحرف التي تستخدم في طابعات الحبر (بدون مسافات، بدون تحكم)</td><td>[:graph:]</td></tr>
<tr><td>[a-z]</td><td>أحرف في حالة صغيرة</td><td>[:lower:]</td></tr>
<tr><td>[\t\x20-\xFF]</td><td>حرف قابل للطباعة (فئة الرسم البياني بالإضافة إلى المسافة وعلامة التبويب)</td><td>[:print:]</td></tr>
<tr><td>[-!"#$%&'()*+,./:;<=>?@[\\\]^_'{|}~]</td><td>أي حرف ترقيم، مثل النقطة (.) والفاصلة المنقوطة (;)</td><td>[:punct:]</td></tr>
<tr><td>[\n\r\t \x0B]</td><td>مسافة بيضاء (سطر جديد، مسافة جدول، مسافة، مسافة جدول أفقي، carriage return)</td><td>[:space:]</td></tr>
<tr><td>[A-Z]</td><td>حروف بحالة كبيرة</td><td>[:upper:]</td></tr>
</table>

| الفئة | الوصف | توسيع |
|---|---|---|
| [:xdigit:] | Hexadecimal digit | [0-9a-fA-F] |
| \s | مسافة بيضاء | [\r\n \t] |
| \S | بدون مسافة بيضاء | [^\r\n \t] |
| \w | Word (identifier) character | [0-9A-Za-z_] |
| \W | Nonword (identifier) character | [^0-9A-Za-z_] |
| \d | Digit | [0-9] |
| \D | Nondigit | [^0-9] |

## المراسي Anchors

يحد المرساة من تطابق موقع معين في السلسلة (لا تتطابق نقاط الارتساء مع الأحرف الفعلية في السلسلة الهدف). يسرد الجدول 4-8 نقاط الارتساء التي تدعمها التعبيرات النمطية.

الجدول 4-8. المراسي

| Anchor | Matches |
| --- | --- |
| ^ | Start of string بداية السلسلة |
| $ | نهاية السلسلة |
| [[:<:]] | بداية الكلمة |
| [[:>:]] | نهاية الكلمة |
| \b | حدود الكلمة (بين \w و \W أو في بداية السلسلة أو نهايتها) |
| \B | Nonword boundary (between \w and \w, or \W and \W) |
| \A | إبتداء السلسلة Beginning of string |
| \Z | نهاية السلسلة أو قبلها \n في النهاية |

| Anchor | Matches |
| --- | --- |
| \z | نهاية السلسلة |
| ^ | بداية السطر (أو بعد \n إذا علامة /m ممكنة) |
| $ | نهاية السطر (أو قبل \n إذا علامة /m ممكنة) |

يتم تعريف حدود الكلمة على أنها النقطة بين حرف مسافة بيضاء وحرف معرف (أبجدي رقمي أو شرطة سفلية):

```
preg_match("/[[:<:]]gun[[:>:]]/", "the Burgundy exploded"); // returns false
preg_match("/gun/", "the Burgundy exploded"); // returns true
```

لاحظ أن بداية السلسلة النصية ونهايتها يمكن اعتبارها حدودًا للكلمات.

### Quantifiers and Greed

عادة ما تكون محددات كمية التعبير العادي جشعة "*greedy*". أي عند مواجهة مُحدِّد كمي "quantifier"، فإن المحرك يطابق قدر الإمكان مع استمرار إرضاء بقية النمط. على سبيل المثال:

```
preg_match("/(<.*>)/", "do <b>not</b> press the button", $match);
// $match[1] is '<b>not</b>'
```

يتطابق التعبير العادي من أول علامة أقل من إلى علامة أكبر من الأخيرة. في الواقع، يتطابق *. مع كل شيء بعد أول علامة أقل من، ويتراجع المحرك لجعله يتطابق بشكل أقل وأقل حتى تظهر في النهاية علامة أكبر من المطلوب مطابقتها.

يمكن أن يكون هذا الجشع مشكلة. في بعض الأحيان تحتاج إلى الحد الأدنى من المطابقة (غير المتشددة) *minimal* (*nongreedy*) *matching* - أي المحددات الكمية التي تطابق أقل عدد ممكن من المرات لإرضاء بقية النمط. تقدم Perl مجموعة متوازية من المحددات الكمية التي تطابق الحد الأدنى. من السهل تذكرها، لأنها مماثلة للمحدِدات الكمية الجشعة، ولكن مع إلحاق علامة استفهام (؟). يوضح الجدول 4-9 المحددات الكمية الجشعة وغير الجشعة المقابلة التي تدعمها التعبيرات العادية على غرار Perl.

الجدول 4-9. محددات الكم الجشعة وغير الجشعة في التعبيرات العادية المتوافقة مع لغة Perl

| الكم الغير الجشع | الكم الجشع |
|---|---|
| ?? | ? |
| *? | * |
| +? | + |
| {m}? | {m} |
| {m,}? | {m,} |

| الكم الغير الجشع | الكم الجشع |
| --- | --- |
| {m,n}? | {m,n} |

إليك كيفية مطابقة علامة باستخدام مُحدد كمي غير جشع:

```
preg_match("/(<.*?>)/", "do <b>not</b> press the button", $match);
// $match[1] is "<b>"
```

هناك طريقة أخرى أسرع وهي استخدام فئة حرف لمطابقة كل حرف ليس أكبر من حتى علامة أكبر من التالية:

```
preg_match("/(<[^>]*>)/", "do <b>not</b> press the button", $match);
// $match[1] is '<b>'
```

### المجموعات غير الملتقطة Noncapturing Groups

إذا قمت بإحاطة جزء من نمط ما بين أقواس، فسيتم التقاط النص المطابق لهذا النمط الفرعي ويمكن الوصول إليه لاحقًا. ومع ذلك، قد ترغب أحيانًا في إنشاء نمط فرعي بدون التقاط النص المطابق. في التعبيرات العادية المتوافقة مع Perl، يمكنك القيام بذلك باستخدام بناء ( *subpattern* ?:):

```
preg_match("/(?:ello)(.*)/", "jello biafra", $match);
// $match[1] is " biafra"
```

## مرجع خلفي Backreferences

يمكنك الإشارة إلى النص الذي تم التقاطه مسبقًا في نمط ذي مرجع خلفي *backreference*: يشير 1\ إلى محتويات النمط الفرعي الأول، ويشير 2\ إلى الثاني، وهكذا. إذا أدخلت أنماطًا فرعية متداخلة، يبدأ الأول بقوس الفتح الأول، ويبدأ الثاني بقوس الفتح الثاني، وهكذا.

على سبيل المثال، هذا يحدد الكلمات المضاعفة:

```
preg_match("/([[:alpha:]]+)\s+\1/", "Paris in the the spring", $m);
// returns true and $m[1] is "the"
```

تلتقط الدالة ()preg_match 99 نمطًا فرعيًا على الأكثر، يتم تجاهل الأنماط الفرعية بعد 99.

## خيارات زائدة Trailing Options

تتيح لك التعبيرات العادية بنمط Perl وضع خيارات أحادية الأحرف (علامات flags) بعد نمط التعبير العادي لتعديل تفسير أو سلوك المطابقة. على سبيل المثال، لمطابقة حالة الأحرف، ما عليك سوى استخدام علامة i:

```
preg_match("/cat/i", "Stop, Catherine!"); // returns true
```

يوضح الجدول 4-10 أي معدّلات Perl مدعومة في التعبيرات العادية المتوافقة مع Perl.

الجدول 4-10. علامات بيرل

| Modifier | المعنى |
|---|---|
| /*regexp*/i | تطابق مع حالة الأحرف |
| /*regexp*/s | جعل النقطة (.) تطابق أي حرف، بما في ذلك السطر الجديد \n |
| /*regexp*/x | حذف المسافات البيضاء والتعليقات من النمط |
| /*regexp*/m | جعل علامة الإقحام (^) تتطابق مع علامة الدولار من قبل، والأسطر الجديدة الداخلية (\n) |
| /*regexp*/e | إذا كانت سلسلة الاستبدال عبارة عن كود PHP، قم بـ ()eval للحصول على سلسلة الاستبدال الفعلية |

تدعم دوال التعبير العادي المتوافقة مع Perl في PHP أيضًا المُعدِّلات الأخرى التي لا تدعمها Perl، كما هو مُدرج في الجدول 4-11.

الجدول 4-11. أعلام PHP إضافية

| Modifier | المعنى |
|---|---|
| /*regexp*/U | يعكس جشع النمط الفرعي ؛ * و + الآن يتطابقان بأقل قدر ممكن، بدلاً من أكبر قدر ممكن |

| Modifier | المعنى |
|---|---|
| */regexp/*u | يتسبب في معالجة سلاسل النمط على أنها UTF-8 |
| */regexp/*X | يتسبب في شرطة مائلة للخلف متبوعة بحرف ليس له معنى خاص لإرسال خطأ |
| */regexp/*A | يتسبب في إرساء بداية السلسلة كما لو كان الحرف الأول من النمط ^ |
| */regexp/*D | يتسبب في تطابق الحرف $ فقط في نهاية السطر |
| */regexp/*S | يتسبب في قيام المحلل اللغوي للتعبير بفحص بنية النموذج بعناية أكبر، لذلك قد يعمل بشكل أسرع قليلاً في المرة التالية (على سبيل المثال في حلقة) |

من الممكن استخدام أكثر من خيار في نمط واحد، كما هو موضح في المثال التالي:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: pay up

Pay me or else!
END;

preg_match("/^subject: (.*)/im", $message, $match);
```

```
print_r($match);

// output: Array ( [0] => Subject: pay up [1] => pay up )
```

### خيارات مضمنة

بالإضافة إلى تحديد خيارات على مستوى النموذج بعد محدد نمط الإغلاق، يمكنك تحديد خيارات ضمن نمط لجعلها تنطبق فقط على جزء من النموذج. بناء الجملة لهذا هو:

```
(?flags:subpattern)
```

على سبيل المثال، كلمة "PHP" هي فقط غير حساسة لحالة الأحرف في هذا المثال:

```
echo preg_match('/I like (?i:PHP)/', 'I like pHp', $match);
print_r($match) ;
// returns true (echo: 1)
// $match[0] is 'I like pHp'
```

يمكن تطبيق خيارات i و m و s و U و x و X داخلياً بهذه الطريقة. يمكنك استخدام خيارات متعددة في وقت واحد:

```
preg_match('/eat (?ix:foo d)/', 'eat FoOD'); // returns true
```

بادئة خيار بشرطة (-) لإيقاف تشغيله:

```
echo preg_match('/I like (?-i:PHP)/', 'I like pHp', $match);
print_r($matche) ;
// returns false (echo: 0)
// $match[0] is ''
```

نموذج بديل يمكّن العلامات أو يعطلها حتى نهاية النمط أو النمط الفرعي المرفق:

```
preg_match('/I like (?i)PHP/', 'I like pHp'); // returns true
preg_match('/I (like (?i)PHP) a lot/', 'I like pHp a lot', $match);
// $match[1] is 'like pHp'
```

العلامات المضمنة لا تمكّن الالتقاط. تحتاج إلى مجموعة إضافية من أقواس الالتقاط للقيام بذلك.

## Lookahead و Lookbehind

في الأنماط، من المفيد أحيانًا أن تكون قادرًا على قول "تطابق هنا إذا كان هذا هو التالي". هذا شائع بشكل خاص عندما تقوم بفصل سلسلة. يصف التعبير العادي الفاصل الذي لم يتم إرجاعه. يمكنك استخدام *lookahead* للتأكد (بدون مطابقته، وبالتالي منع إرجاعه) من وجود المزيد من البيانات بعد الفاصل. وبالمثل، يتحقق *lookbehind* من النص السابق.

تأتي Lookahead و lookbehind في شكلين: إيجابي وسلبي ”*positive* and *negative*“. يقول الشكل الإيجابي لـ Lookahead أو lookbehind "يجب أن يكون النص التالي/السابق هكذا." يشير الشكل السلبي لـ Lookahead أو lookbehind "يجب ألا يكون النص التالي/السابق هكذا." يوضح الجدول 4-12 التركيبات الأربعة التي يمكنك استخدامها في الأنماط المتوافقة مع Perl. لا يلتقط أي من هذه التركيبات النص.

الجدول 4-12. Lookahead و lookbehind التأكيدات

| المعنى | البنية |
| --- | --- |
| Positive lookahead | (?=*subpattern*) |
| Negative lookahead | (?!*subpattern*) |
| Positive lookbehind | (?<=*subpattern*) |
| Negative lookbehind | (?<!*subpattern*) |

أحد الاستخدامات البسيطة لميزة lookahead الإيجابي هو تقسيم ملف بريد Unix mbox إلى رسائل فردية. تشير الكلمة "From" التي تبدأ سطراً بمفردها إلى بداية رسالة جديدة، لذلك يمكنك تقسيم صندوق البريد إلى رسائل عن طريق تحديد الفاصل كنقطة يكون فيها النص التالي "From" في بداية السطر:

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

من الاستخدامات البسيطة للبحث الخلفي السلبي "negative lookbehind" استخراج السلاسل المقتبسة التي تحتوي على محددات بين علامات اقتباس. على سبيل المثال، إليك كيفية استخراج سلسلة ذات علامات اقتباس مفردة (لاحظ أنه يتم التعليق على التعبير العادي باستخدام معدل x):

```
$input = <<< END
name = 'Tim O\'Reilly';
END;

$pattern = <<< END
' # opening quote
( # begin capturing
 .*? # the string
 (?<! \\\\ ) # skip escaped quotes
) # end capturing
' # closing quote
END;
preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O\'Reilly
```

الجزء الصعب الوحيد هو أنه للحصول على نمط ينظر إلى الخلف لمعرفة ما إذا كان الحرف الأخير عبارة عن شرطة مائلة للخلف، نحتاج إلى الهروب من الشرطة المائلة للخلف لمنع محرك التعبير العادي من رؤية \)، مما يعني وجود قوس قريب حرفيًا. بمعنى آخر، علينا استخدام الشرطة المائلة للخلف مرتين: \\). لكن قواعد اقتباس السلسلة في PHP تنص على أن \\ ينتج شرطة مائلة واحدة عكسية، لذلك ينتهي بنا الأمر بطلب أربع شرطات مائلة للخلف للحصول على واحدة من خلال التعبير العادي! هذا هو السبب في أن التعبيرات النمطية تشتهر بصعوبة قراءتها.

تحد Perl lookbehind من التعبيرات ذات العرض الثابت. أي أن التعبيرات لا يمكن أن تحتوي على محددات كمية، وإذا كنت تستخدم التناوب "alternation"، فيجب أن تكون جميع الاختيارات بنفس الطول. يحظر محرك التعبير العادي المتوافق مع Perl أيضًا المحددات الكمية في البحث الخلفي، ولكنه يسمح ببدائل ذات أطوال مختلفة.

## قطع Cut

يمنع النمط الفرعي، أو cut، الذي نادرًا ما يستخدم مرة واحدة فقط، السلوك الأسوأ لمحرك التعبير العادي في بعض أنواع الأنماط. لا يتم أبدًا التراجع عن النمط الفرعي مرة واحدة.

الاستخدام الشائع للنمط الفرعي الوحيد هو عندما يكون لديك تعبير متكرر قد يتكرر في حد ذاته:

```
/(a+|b+)*\.+/
```

يستغرق مقتطف الشفرة هذا عدة ثوانٍ للإبلاغ عن الفشل:

```
$p = '/(a+|b+)*\.+$/';
$s                                                    =
'ababababbabbbabbaaaaabbbbabbababababababbba..!';

if (preg_match($p, $s)) {
 echo "Y";
}
else {
 echo "N";
```

```
}
```

هذا لأن محرك التعبير العادي يحاول بدء التطابق في جميع الأماكن المختلفة، ولكن عليه التراجع عن كل مكان، الأمر الذي يستغرق وقتًا. إذا كنت تعلم أنه بمجرد مطابقة شيء ما، فلا يجب التراجع عنه مطلقًا، فيجب عليك وضع علامة (*?>subpattern*):

```
$p = '/(?>a+|b+)*\.+$/';
```

القطع لا يغير نتيجة المطابقة أبدًا، إنه ببساطة يجعله يفشل بشكل أسرع.

## التعبيرات الشرطية

يشبه التعبير الشرطي عبارة if في تعبير عادي. الشكل العام هو:

```
(?(condition)yespattern)
(?(condition)yespattern|nopattern)
```

إذا نجح التأكيد، فإن محرك التعبير العادي يطابق *yespattern*. مع النموذج الثاني، إذا لم ينجح التأكيد، يتخطى محرك التعبير العادي *yespattern* ويحاول مطابقة nopattern.

يمكن أن يكون التأكيد أحد نوعين: إما مرجع خلفي "backreference" أو تطابق متقدم "lookahead" أو تطابق خلفي "lookbehind". للإشارة إلى سلسلة فرعية تمت مطابقتها مسبقًا، يكون

التأكيد رقمًا من 1 إلى 99 (معظم الترجعات الخلفية المتاحة). يستخدم الشرط النمط الموجود في التوكيد فقط في حالة مطابقة المرجع الخلفي. إذا لم يكن التأكيد مرجعًا خلفيًا، فيجب أن يكون تأكيدًا إيجابيًا أو سلبيًا lookahead or lookbehind.

## الدوال

هناك خمس فئات من الدوال التي تعمل مع التعبيرات العادية المتوافقة مع Perl: المطابقة matching، والاستبدال replacing، والتقسيم splitting، والتصفية filtering، والدالة المساعدة لاقتباس النص.

## التطابق

تقوم الدالة ()preg_match بتنفيذ مطابقة نمط Perl على سلسلة. إنه مكافئ لعامل //m عامل في Perl. تأخذ الدالة ()preg_match_all نفس المدخلات وتعطي نفس القيمة المرجعة للدالة ()preg_match، باستثناء أنها تأخذ نمط Perl بدلاً من نمط قياسي:

```
$found = preg_match(pattern, string [, captured ]);
```

كمثال:

```
preg_match('/y.*e$/', 'Sylvie'); // returns true
preg_match('/y(.*)e$/',  'Sylvie',  $m);  //  $m  is
array('ylvie', 'lvi')
```

على الرغم من وجود دالة ()preg_match لمطابقة حالة الأحرف غير الحساسة، فلا توجد دالة ()preg_matchi. بدلاً من ذلك، استخدم علامة i على النمط:

```
preg_match('y.*e$/i', 'SyLvIe'); // returns true
```

تطابق دالة preg_match_all() بشكل متكرر من حيث انتهت المطابقة الأخيرة، حتى لا يمكن إجراء المزيد من التطابقات:

```
$found = preg_match_all(pattern, string, matches [, order ]);
```

تحدد قيمة *order*، إما PREG_PATTERN_ORDER أو PREG_SET_ORDER، مخطط التطابقات. سنلقي نظرة على كليهما، باستخدام هذا الكود كدليل:

```
$string = <<< END
13 dogs
12 rabbits
8 cows
1 goat
END;
preg_match_all('/(\d+) (\S+)/', $string, $m1, PREG_PATTERN_ORDER);
preg_match_all('/(\d+) (\S+)/', $string, $m2, PREG_SET_ORDER);
```

باستخدام PREG_PATTERN_ORDER (الافتراضي)، يتوافق كل عنصر في المصفوفة مع نمط التقاط فرعي معين. إذن، $m1[0] عبارة عن مصفوفة من جميع السلاسل الفرعية التي تطابق النمط، و $m1[1] عبارة عن مصفوفة من جميع السلاسل الفرعية التي تطابق النمط الفرعي الأول (الأرقام)، و

$m1[2] عبارة عن مصفوفة من كل السلاسل الفرعية التي تطابق النمط الفرعي الثاني (الكلمات). تحتوي المصفوفة $m1 على عنصر واحد أكثر من الأنماط الفرعية.

باستخدام PREG_SET_ORDER، يتوافق كل عنصر من عناصر المصفوفة مع المحاولة التالية لمطابقة النمط بأكمله. لذا فإن $m2[0] عبارة عن مصفوفة من المجموعة الأولى من التطابقات ("13 dogs" ، "13" ، "dogs")، $m2[1] هي مصفوفة من المجموعة الثانية من التطابقات ("12 rabbits" ، "12" ، "rabbits") وهكذا. تحتوي المصفوفة $m2 على العديد من العناصر حيث كانت هناك تطابقات ناجحة للنمط بأكمله.

المثال 4-1 يجلب HTML في عنوان ويب معين إلى سلسلة ويستخرج عناوين URL من HTML. لكل عنوان URL، يقوم بإنشاء ارتباط إلى البرنامج الذي سيعرض عناوين URL على هذا العنوان.

مثال 4-1. استخراج عناوين المواقع من صفحة HTML

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
 $url = $_POST['url'];
}
else {
 $url = $_GET['url'];
}
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
```

```
 <p>URL: <input type="text" name="url" value="<?php
echo $url ?>" /><br />
 <input type="submit">
</form>

<?php
if ($url) {
 $remote = fopen($url, 'r'); {
 $html = fread($remote, 1048576); // read up to 1 MB of
HTML
 }
 fclose($remote);

 $urls = '(http|telnet|gopher|file|wais|ftp)';
 $ltrs = '\w';
 $gunk = '/#~:.?+=&%@!\-';
 $punc = '.:?\-';
 $any = "{$ltrs}{$gunk}{$punc}";

 preg_match_all("{
 \b # start at word boundary
 {$urls}: # need resource and a colon
 [{$any}] +? # followed by one or more of any valid
 # characters—but be conservative
 # and take only what you need
 (?= # the match ends at
 [{$punc}]* # punctuation
```

```
 [^{$any}] # followed by a non-URL character
 | # or
 \$ # the end of the string
 )
 }x", $html, $matches);

 printf("I found %d URLs<P>\n", sizeof($matches[0]));

 foreach ($matches[0] as $u) {
 $link = $_SERVER['PHP_SELF'] . '?url=' . urlencode($u);
 echo "<a href=\"{$link}\">{$u}</a><br />\n";
 }
}
```

استبدال

تتصرف الدالة ()preg_replace مثل عملية البحث والاستبدال في محرر النصوص. يعثر على كل تكرارات النمط في سلسلة ويغير تلك التكرارات إلى شيء آخر:

```
$new = preg_replace(pattern, replacement, subject [, limit ]);
```

الاستخدام الأكثر شيوعًا له كل سلاسل المدخلات باستثناء حد العدد الصحيح integer *limit*. الحد هو الحد الأقصى لعدد تكرارات النمط المراد استبداله (الافتراضي، والسلوك عند تجاوز حد 1-، هو كل التكرارات):

```
$better = preg_replace('/<.*?>/', '!', 'do <b>not</b> press the button');
```

```
// $better is 'do !not! press the button'
```

مرر مجموعة من السلاسل كموضوع *subject* لإجراء الاستبدال عليها جميعًا. يتم إرجاع السلاسل الجديدة من ()preg_replace:

```
$names = array('Fred Flintstone',
 'Barney Rubble',
 'Wilma Flintstone',
 'Betty Rubble');
$tidy = preg_replace('/(\w)\w* (\w+)/', '\1 \2', $names);
// $tidy is array ('F Flintstone', 'B Rubble', 'W Flintstone', 'B Rubble')
```

لإجراء استبدالات متعددة على نفس السلسلة أو مجموعة السلاسل النصية باستدعاء واحد لـ ()preg_replace، قم بتمرير مصفوفات من الأنماط والاستبدالات:

```
$contractions = array("/don't/i", "/won't/i", "/can't/i");
$expansions = array('do not', 'will not', 'can not');
$string = "Please don't yell - I can't jump while you won't speak";
$longer = preg_replace($contractions, $expansions, $string);
// $longer is 'Please do not yell - I can not jump while you will not speak';
```

إذا أعطيت استبدالات أقل من الأنماط، فسيتم حذف النص المطابق للأنماط الإضافية. هذه طريقة سهلة لحذف الكثير من الأشياء مرة واحدة:

```
$htmlGunk = array('/<.*?>/', '/&.*?;/');
$html = '&eacute; : <b>very</b> cute';
$stripped = preg_replace($htmlGunk, array(), $html);
// $stripped is ' : very cute'
```

إذا أعطيت مصفوفة من الأنماط لكن مع استبدال سلسلة واحدة، فسيتم استخدام نفس الاستبدال لكل نمط:

```
$stripped = preg_replace($htmlGunk, '', $html);
```

يمكن أن يستخدم التبديل backreferences. على عكس backreferences في الأنماط، فإن الصيغة المفضلة لـ backreferences في التبديلات هي $1 و $2 و $3 وما إلى ذلك. فمثلا:

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Fred
Flintstone')
Flintstone, F.
```

المُعدِّل e/ يجعل ()preg_replace يتعامل مع سلسلة الاستبدال ككود PHP الذي يُعيد السلسلة الفعلية لاستخدامها في الاستبدال. على سبيل المثال، يؤدي هذا إلى تحويل كل درجة حرارة مئوية إلى فهرنهايت:

```
$string = 'It was 5C outside, 20C inside';
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32', $string);
It was 41 outside, 68 inside
```

هذا المثال الأكثر تعقيدًا يوسع المتغيرات في سلسلة:

```
$name = 'Fred';
$age = 35;
$string = '$name is $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

كل تطابق يعزل اسم المتغير ($name, $age). يشير $1 في البديل إلى هذه الأسماء، لذا فإن كود PHP الذي تم تنفيذه بالفعل هو $name و $age. يتم تقييم هذا الكود إلى قيمة المتغير، وهو ما يتم استخدامه كبديل. يا للعجب!

الاختلاف في preg_replace() هو preg_replace_callback(). هذا يستدعي دالة للحصول على السلسلة البديلة. تمرر الدالة مصفوفة من التطابقات (العنصر الصفري هو كل النص المطابق للنمط، الأول هو محتويات النمط الفرعي الأول الملتقط، وهكذا). فمثلا:

```
function titlecase($s)
{
 return ucfirst(strtolower($s[0]));
}

$string = 'goodbye cruel world';
$new = preg_replace_callback('/\w+/', 'titlecase',
$string);
echo $new;

Goodbye Cruel World
```

## شق spliting

بينما تستخدم ()preg_match_all لاستخراج أجزاء من سلسلة عندما تعرف ما هي تلك القطع، استخدم ()preg_split لاستخراج القطع عندما تعرف ما الذي يفصل القطع عن بعضها البعض:

```
$chunks = preg_split(pattern, string [, limit [, flags
]]);
```

يتطابق النمط مع فاصل بين قطعتين. بشكل افتراضي، لا يتم إرجاع الفواصل. يحدد الحد الاختياري الحد الأقصى لعدد القطع المراد إرجاعها (1- هو الافتراضي، مما يعني جميع الأجزاء). مدخل العلامات هي عبارة عن مزيج أحادي الاتجاه من علامات PREG_SPLIT_NO_EMPTY (لم يتم إرجاع الأجزاء الفارغة) و PREG_SPLIT_DELIM_CAPTURE (يتم إرجاع أجزاء من السلسلة الملتقطة في النموذج).

على سبيل المثال، لاستخراج المعاملات فقط من تعبير رقمي بسيط، استخدم:

```
$ops = preg_split('{[+*/-]}', '3+5*9/2');
// $ops is array('3', '5', '9', '2')
```

لاستخراج المعاملات والعمليات، استخدم:

```
$ops   =   preg_split('{([+*/-])}',   '3+5*9/2',   -1,
PREG_SPLIT_DELIM_CAPTURE);
// $ops is array('3', '+', '5', '*', '9', '/', '2')
```

يتطابق نمط فارغ عند كل حد بين الأحرف في السلسلة وفي بداية السلسلة ونهايتها. يتيح لك ذلك تقسيم سلسلة إلى مجموعة من الأحرف:

```
$array = preg_split('//', $string);
```

تصفية مصفوفة بتعبير منتظم

تُرجع الدالة preg_grep() عناصر المصفوفة التي تطابق نمطًا معيناً:

```
$matching = preg_grep(pattern, array);
```

على سبيل المثال، للحصول على أسماء الملفات التي تنتهي بـ txt. فقط، استخدم:

```
$textfiles = preg_grep('/\.txt$/', $filenames);
```

الاقتباس عن العبارات العادية

تنشئ الدالة preg_quote() تعبيراً عادياً يطابق سلسلة نصية معينة فقط:

```
$re = preg_quote(string [, delimiter ]);
```

كل حرف في سلسلة لها معنى خاص داخل تعبير عادي (على سبيل المثال، * أو $) يتم تهيئته بشرطة مائلة للخلف:

```
echo preg_quote('$5.00 (five bucks)');
\$5\.00 \(five bucks\)
```

المدخل الثاني الاختياري هو حرف إضافي يتم اقتباسه. عادة، تقوم بتمرير محدد التعبير النمطي الخاص بك هنا:

```
$toFind = '/usr/local/etc/rsync.conf';
$re = preg_quote($toFind, '/');

if (preg_match("/{$re}/", $filename)) {
 // found it!
}
```

## الاختلافات عن تعبيرات Perl العادية

على الرغم من تشابهها الشديد، إلا أن تنفيذ PHP للتعبيرات العادية بأسلوب Perl له بعض الاختلافات الطفيفة عن التعبيرات العادية الفعلية لـ Perl:

- ❖ لا يُسمح بالحرف الفارغ (ASCII 0) كحرف حرفي ضمن سلسلة نمط. يمكنك الرجوع إليه بطرق أخرى (\000، \x00، إلخ).
- ❖ الخيارات \E و \G و \L و \l و \Q و \u و \U غير مدعومة.
- ❖ البنية (?{ *some perl code* }) غير مدعومة.
- ❖ يتم اعتماد المعدّلات /D و /G و /U و /u و /A و /X.
- ❖ يتم احتساب علامة التبويب العمودية \v كحرف مسافة بيضاء.
- ❖ لا يمكن تكرار تأكيدات Lookahead و lookbehind باستخدام * أو + أو؟.
- ❖ لا يتم تذكر عمليات الإرسال المقوسة داخل التأكيدات السلبية.
- ❖ يمكن أن تكون فروع التناوب داخل تأكيد lookbehind ذات أطوال مختلفة.

# ماذا بعد

الآن بعد أن عرفت كل شيء يجب أن تعرفه عن السلاسل والعمل معها، فإن الجزء الرئيسي التالي من PHP الذي سنركز عليه هو المصفوفات. ستتحداك أنواع البيانات المركبة هذه، ولكن عليك أن تتعرف عليها جيدًا، حيث تعمل PHP معها في العديد من المجالات. تعلم كيفية إضافة عناصر المصفوفات وفرز المصفوفات والتعامل مع الأشكال متعددة الأبعاد من المصفوفات أمر ضروري لكونك مطور PHP جيد.

# الفصل الخامس: المصوفات

كما ناقشنا في الفصل الثاني، تدعم PHP كلاً من أنواع البيانات العددية "scalar" والمركبة "compound". في هذا الفصل، سنناقش أحد أنواع المركبات: المصفوفات.

**المصفوفة:** هي مجموعة من قيم البيانات المنظمة كمجموعة مرتبة من أزواج المفتاح والقيمة "key-value". قد يكون من المفيد التفكير في المصفوفة، مثل كرتونة البيض. يمكن لكل حجرة من كرتونة البيض أن تحمل بيضة، لكنها تتحرك كوعاء واحد شامل. ومثلما لا يجب أن يحتوي كرتونة البيض على البيض فقط (يمكنك وضع أي شيء فيه، مثل: الصخور، أو كرات الثلج، أو البرسيم رباعي الأوراق، أو الصواميل والمسامير)، لذا فإن المصفوفة أيضًا لا تقتصر على نوع واحد من البيانات. يمكن أن تحتوي على سلاسل وأعداد صحيحة وقيم منطقية وما إلى ذلك. بالإضافة إلى ذلك، يمكن أن تحتوي حجرة المصفوفات أيضًا على مصفوفات أخرى.

يتحدث هذا الفصل عن إنشاء مصفوفة وإضافة عناصر وإزالتها منها والتكرار فوق محتويات المصفوفة. نظرًا لأن المصفوفات شائعة جدًا ومفيدة، فهناك العديد من الدوال المضمنة التي تعمل معها في PHP. على سبيل المثال، إذا كنت ترغب في إرسال بريد إلكتروني إلى أكثر من عنوان بريد إلكتروني، فسوف تقوم بتخزين عناوين البريد الإلكتروني في مصفوفة ثم تقوم بالتكرار خلال المصفوفة، وإرسال الرسالة إلى عنوان البريد الإلكتروني الحالي. أيضًا، إذا كان لديك نموذج يسمح بتحديدات متعددة، فسيتم إرجاع العناصر التي حددها المستخدم في مصفوفة.

# المصفوفات المفهرسة مقابل المصفوفة الترابطية

**Indexed Versus Associative Arrays**

يوجد نوعان من المصفوفات في PHP: مفهرسة "indexed" وترابطية "associative". مفاتيح المصفوفة المفهرسة هي أعداد صحيحة تبدأ بالرقم 0. تستخدم المصفوفات المفهرسة عندما تحدد الأشياء حسب موقعها. تحتوي المصفوفات الترابطية على سلاسل كمفاتيح وتتصرف مثل الجداول المكونة من عمودين. العمود الأول هو المفتاح، والذي يستخدم للوصول إلى القيمة.

تقوم PHP بتخزين جميع المصفوفات داخلياً كمصفوفات ترابطية؛ الفرق الوحيد بين المصفوفات الترابطية والمفهرسة هو ماهية المفاتيح. يتم توفير بعض ميزات المصفوفات بشكل أساسي للاستخدام مع المصفوفات المفهرسة لأنها تفترض أن لديك أو تريد مفاتيح تكون أعدادًا صحيحة متتالية تبدأ من 0. في كلتا الحالتين، تكون المفاتيح فريدة. بمعنى آخر، لا يمكن أن يكون لديك عنصران بنفس المفتاح، بغض النظر عما إذا كان المفتاح سلسلة أو عددًا صحيحًا.

تحتوي مصفوفات PHP على ترتيب داخلي لعناصرها يكون مستقلاً عن المفاتيح والقيم، وهناك دوال يمكنك استخدامها لاجتياز المصفوفات بناءً على هذا الترتيب الداخلي. عادةً ما يكون الترتيب هو الترتيب الذي تم فيه إدراج القيم في المصفوفة، لكن دوال الفرز الموضحة لاحقًا في هذا الفصل تتيح لك تغيير الترتيب إلى ترتيب بناءً على المفاتيح أو القيم أو أي شيء آخر تختاره.

# تحديد عناصر المصفوفة

قبل أن ننظر في إنشاء مصفوفة، دعنا نلقي نظرة على هيكل مصفوفة موجودة. يمكنك الوصول إلى قيم محددة من مصفوفة موجودة باستخدام اسم متغير المصفوفة، متبوعًا بمفتاح العنصر، أو الفهرس، داخل أقواس مربعة:

```
$age['fred']
$shows[2]
```

يمكن أن يكون المفتاح إما سلسلة أو عددًا صحيحًا. يتم التعامل مع قيم السلسلة المكافئة للأرقام الصحيحة (بدون الأصفار البادئة) كأعداد صحيحة. وبالتالي، يشير $array[3] و $array['3'] إلى نفس العنصر، بينما يشير $array['03'] إلى عنصر مختلف. الأرقام السالبة هي مفاتيح صالحة، لكنها لا تحدد مواضع من نهاية المصفوفة كما هو الحال في Perl.

ليس عليك اقتباس سلاسل من كلمة واحدة. على سبيل المثال، $age['fred'] هي نفسها $age[fred]. ومع ذلك، يُعتبر استخدام علامات الاقتباس دائمًا أسلوب PHP جيدًا، لأن المفاتيح التي لا تحتوي على أي مسافة لا يمكن تمييزها عن الثوابت. عندما تستخدم ثابتًا كمؤشر بدون علامات تنصيص، تستخدم PHP قيمة الثابت كمؤشر وتصدر تحذيرًا. سيؤدي هذا إلى ظهور خطأ في الإصدارات المستقبلية من PHP:

```
$person = array("name" => 'Peter');
print "Hello, {$person[name]}";
// output: Hello, Peter
// this 'works' but emits this warning as well:
```

```
Warning: Use of undefined constant name - assumed 'name' (this will throw an
Error in a future version of PHP)
```

يجب عليك استخدام علامات الاقتباس إذا كنت تستخدم الاستيفاء "interpolation" لإنشاء فهرس المصفوفة:

```
$person = array("name" => 'Peter');
print "Hello, {$person["name"]}";// output: Hello, Peter (with no warning)
```

على الرغم من أنه اختياري تقنياً، إلا أنه يجب عليك أيضًا اقتباس المفتاح إذا كنت تستكمل بحث مصفوفة للتأكد من حصولك على القيمة التي تتوقعها. ضع في اعتبارك هذا المثال:

```
define('NAME', 'bob');
$person = array("name" => 'Peter');
echo "Hello, {$person['name']}";
echo "<br/>" ;
echo "Hello, NAME";
echo "<br/>" ;
echo NAME ;
// output:
Hello, Peter
Hello, NAME
bob
```

# تخزين البيانات في المصفوفات

سيؤدي تخزين قيمة في مصفوفة إلى إنشاء المصفوفة إذا لم تكن موجودة بالفعل، ولكن محاولة استرداد قيمة من مصفوفة لم يتم تعريفها لن تنشئ المصفوفة. فمثلا:

```
// $addresses not defined before this point
echo $addresses[0]; // prints nothing
echo $addresses; // prints nothing

$addresses[0] = "spam@cyberpromo.net";
echo $addresses; // prints "Array"
```

يمكن أن يؤدي استخدام مهمة بسيطة لتهيئة مصفوفة في برنامجك إلى كود مثل هذا:

```
$addresses[0] = "spam@cyberpromo.net";
$addresses[1] = "abuse@example.com";
$addresses[2] = "root@example.com";
```

هذه مصفوفة مفهرسة، تبدأ مؤشرات الأعداد الصحيحة بها عند 0. وإليك مصفوفة ترابطية:

```
$price['gasket'] = 15.29;
$price['wheel'] = 75.25;
$price['tire'] = 50.00;
```

أسهل طريقة لتهيئة المصفوفة هي استخدام بنية array() ، التي تبني مصفوفة من مدخلاتها. يؤدي هذا إلى إنشاء مصفوفة مفهرسة، ويتم إنشاء قيم الفهرس (بدءًا من 0) تلقائيًا:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",
"root@example.com");
```

لإنشاء مصفوفة ترابطية باستخدام array()، استخدم الرمز => لفصل المؤشرات "indices" (المفاتيح) عن القيم:

```
$price = array(
 'gasket' => 15.29,
 'wheel' => 75.25,
 'tire' => 50.00
);
```

لاحظ استخدام المسافة البيضاء والمحاذاة. كان من الممكن أن نجمع الكود، لكنها لم تكن سهلة القراءة (وهذا يعادل نموذج الكود السابق)، أو إضافة القيم أو إزالتها بسهولة:

```
$price = array('gasket' => 15.29, 'wheel' => 75.25,
'tire' => 50.00);
```

يمكنك أيضًا تحديد مصفوفة باستخدام صيغة بديلة أقصر:

```
$price = ['gasket' => 15.29, 'wheel' => 75.25, 'tire'
=> 50.0];
```

لإنشاء مصفوفة فارغة، لا تمرر أي مدخلات إلى ()array:

```
$addresses = array();
```

يمكنك تحديد مفتاح أولي باستخدام => ثم قائمة القيم. يتم إدخال القيم في المصفوفة بدءًا من هذا المفتاح، مع القيم اللاحقة التي تحتوي على مفاتيح متسلسلة:

```
$days = array(1 => "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun");
// 2 is Tue, 3 is Wed, etc.
```

إذا كان الفهرس الأولي عبارة عن سلسلة غير رقمية، فإن المؤشرات اللاحقة هي أعداد صحيحة تبدأ من 0. وبالتالي، من المحتمل أن يكون الكود التالي خطأً:

```
$whoops = array('Fri' => "Black", "Brown", "Green");

// same as
$whoops = array('Fri' => "Black", 0 => "Brown", 1 =>
"Green");
```

### إلحاق القيم بمصفوفة

لإضافة المزيد من القيم إلى نهاية مصفوفة مفهرسة موجودة، استخدم بناء الجملة []:

```
$family = array("Fred", "Wilma");
$family[] = "Pebbles"; // $family[2] is "Pebbles"
```

يفترض هذا البناء أن فهارس المصفوفة هي أرقام وتقوم بتعيين العناصر في الفهرس الرقمي المتاح التالي، بدءًا من 0. محاولة الإلحاق بمصفوفة ترابطية بدون مفاتيح مناسبة غالبًا ما يكون خطأً مبرمجًا، ولكن PHP ستمنح العناصر الجديدة مؤشرات رقمية دون إصدار تحذير:

```
$person = array('name' => "Fred");
$person[] = "Wilma"; // $person[0] is now "Wilma"
```

## تعيين نطاق من القيم

تنشئ الدالة range() مصفوفة من الأعداد الصحيحة المتتالية أو قيم الأحرف بين القيمتين اللتين تمررهما إليها وتضمينهما كمتغيرات. فمثلا:

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z'); // $letters holds the alphabet
$reversedNumbers = range(5, 2); // $reversedNumbers = array(5, 4, 3, 2);
```

يتم استخدام الحرف الأول فقط من مدخل السلسلة لبناء النطاق:

```
range("aaa", "zzz"); // same as range('a','z')
```

## الحصول على حجم المصفوفة

دالتا count() و sizeof() متطابقتان في الاستخدام والتأثير. يعيدون عدد العناصر في المصفوفة. لا يوجد تفضيل أسلوبي حول الدالة التي تستخدمها. هذا مثال:

```
$family = array("Fred", "Wilma", "Pebbles");
```

```
$size = count($family); // $size is 3
```

تحسب هذه الدالة قيم المصفوفة التي تم تعيينها بالفعل فقط:

```
$confusion = array( 10 => "ten", 11 => "eleven", 12 =>
"twelve");
$size = count($confusion); // $size is 3
```

## حشو أو تبطين مصفوفة Padding an Array

لإنشاء مصفوفة بقيم مهيأة لنفس المحتوى، استخدم ()array_pad. المدخل الأولى لـ ()array_pad هو المصفوفة، والمدخل الثاني هو الحد الأدنى لعدد العناصر التي تريد أن تحتوي عليها المصفوفة، والمدخل الثالث هو القيمة لإعطاء أي عناصر تم إنشاؤها. ترجع الدالة ()array_pad مصفوفة مبطنة جديدة، تاركةً مصفوفة المدخل (المصدر) الخاصة بها بمفردها.

إليك عملية ()array_pad:

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded is now
array(5, 10, 0, 0, 0)
```

لاحظ كيف يتم إلحاق القيم الجديدة بالمصفوفة. إذا كنت تريد إضافة القيم الجديدة إلى بداية المصفوفة، فاستخدم المدخل الثاني السالب:

```
$padded = array_pad($scores, -5, 0); // $padded is now
array(0, 0, 0, 5, 10);
```

إذا قمت بتعبئة مصفوفة ترابطية، فسيتم الاحتفاظ بالمفاتيح الموجودة. العناصر الجديدة سيكون لها مفاتيح رقمية تبدأ من 0.

# المصفوفات متعددة الأبعاد

يمكن أن تكون القيم في المصفوفة مصفوفات. يتيح لك ذلك إنشاء مصفوفات متعددة الأبعاد بسهولة:

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
```

يمكنك الرجوع إلى عناصر المصفوفات متعددة الأبعاد بإلحاق المزيد من الأقواس المربعة، []:

```
$value = $multi[2][0]; // row 2, column 0. $value = 7
```

لاستيفاء بحث عن مصفوفة متعددة الأبعاد، يجب عليك إحاطة البحث عن المصفوفة بالكامل بأقواس معقوفة:

```
echo("The value at row 2, column 0 is {$multi[2][0]}\n");
```

يؤدي الفشل في استخدام الأقواس المتعرجة إلى إخراج مثل هذا:

```
The value at row 2, column 0 is Array[0]
```

# استخراج قيم متعددة

لنسخ جميع قيم المصفوفة إلى متغيرات، استخدم بناء ()list:

```
list ($variable, ...) = $array;
```

يتم نسخ قيم المصفوفة إلى المتغيرات المدرجة بالترتيب الداخلي للمصفوفة. هذا هو الترتيب الذي تم إدراجها به افتراضياً، ولكن دوال الفرز الموضحة لاحقًا تتيح لك تغيير ذلك. هذا مثال:

```
$person = array("Fred", 35, "Betty");
list($name, $age, $wife) = $person;
// $name is "Fred", $age is 35, $wife is "Betty"
```

**ملاحظة:**

يعد استخدام دالة ()list ممارسة شائعة لانتقاء القيم من تحديد قاعدة بيانات حيث يتم إرجاع صف واحد فقط. يقوم هذا تلقائيًا بتحميل البيانات من الاستعلام البسيط إلى سلسلة من المتغيرات المحلية. فيما يلي مثال لاختيار فريقين متعارضين من قاعدة بيانات جدولة الألعاب الرياضية:

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE
Ident = 7";
$result = mysql_query($sql);
list($hometeam,                $awayteam)              =
mysql_fetch_assoc($result);
```

هناك تغطية أكبر لقواعد البيانات في الفصل التاسع.

إذا كانت لديك قيم في المصفوفة أكثر من تلك الموجودة في ()list، فسيتم تجاهل القيم الإضافية:

```
$person = array("Fred", 35, "Betty");
list($name, $age) = $person; // $name is "Fred", $age
is 35
```

إذا كانت لديك قيم في ()list أكثر من تلك الموجودة في المصفوفة، فسيتم تعيين القيم الإضافية على NULL:

```
$values = array("hello", "world");
list($a, $b, $c) = $values; // $a is "hello", $b is
"world", $c is NULL
```

فاصلتان متتاليتان أو أكثر في ()list تخطي القيم في المصفوفة:

```
$values = range('a', 'e'); // use range to populate the
array
list($m, , $n, , $o) = $values; // $m is "a", $n is "c",
$o is "e"
```

## تشريح المصفوفة Slicing an Array

لاستخراج مجموعة فرعية فقط من المصفوفة ، استخدم الدالة ()array_slice:

```
$subset = array_slice(array, offset, length);
```

ترجع الدالة ()array_slice مصفوفة جديدة تتكون من سلسلة متتالية من القيم من المصفوفة الأصلية. تحدد معلمة الإزاحة "*offset*" العنصر الأولي المراد نسخه (يمثل 0 العنصر الأول في المصفوفة)، وتحدد

معلمة الطول *length* عدد القيم المراد نسخها. تحتوي المصفوفة الجديدة على مفاتيح رقمية متتالية تبدأ من 0. على سبيل المثال:

```
$people = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
$middle = array_slice($people, 2, 2); // $middle is array("Harriet", "Brenda")
```

بشكل عام، من المفيد فقط استخدام ()array_slice على المصفوفات المفهرسة (أي تلك التي تحتوي على فهارس أعداد صحيحة متتالية تبدأ من 0):

```
// this use of array_slice() makes no sense
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
$subset = array_slice($person, 1, 2); // $subset is array(0 => 35, 1 => "Betty")
```

اجمع ()array_slice مع ()list لاستخراج بعض القيم فقط إلى المتغيرات:

```
$order = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
list($second, $third) = array_slice($order, 1, 2);
// $second is "Dick", $third is "Harriet"
```

## تقسيم المصفوفة إلى أجزاء

لتقسيم مصفوفة إلى مصفوفات أصغر حجمًا، استخدم الدالة ()array_chunk:

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

تقوم الدالة بإرجاع مصفوفة من المصفوفات الأصغر. المدخل الثالث، *preserve_keys* ، هي قيمة منطقية تحدد ما إذا كانت عناصر المصفوفات الجديدة لها نفس المفاتيح الموجودة في الأصل (مفيدة للمصفوفات الترابطية) أو مفاتيح رقمية جديدة تبدأ من 0 (مفيدة للمصفوفات المفهرسة). الإعداد الافتراضي هو تعيين مفاتيح جديدة، كما هو موضح هنا:

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);

Array (
 [0] => Array (
 [0] => 1
 [1] => 2
 [2] => 3
 )
 [1] => Array (
 [0] => 4
 [1] => 5
 [2] => 6
 )
 [2] => Array (
 [0] => 7
 )
)
```

## المفاتيح والقيم

ترجع الدالة array_keys() مصفوفة تتكون فقط من المفاتيح الموجودة في المصفوفة بالترتيب الداخلي:

```
$arrayOfKeys = array_keys(array);
```

هذا مثال:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$keys = array_keys($person); // $keys is array("name", "age", "wife")
```

توفر PHP أيضًا دالة (أقل فائدة بشكل عام) لاسترداد مصفوفة من القيم الموجودة في المصفوفة فقط، array_values():

```
$arrayOfValues = array_values(array);
```

كما هو الحال مع array_keys()، يتم إرجاع القيم بالترتيب الداخلي للمصفوفة:

```
$values = array_values($person); // $values is array("Fred", 35, "Wilma");
```

## التحقق من وجود عنصر

لمعرفة ما إذا كان العنصر موجودًا في المصفوفة، استخدم الدالة array_key_exists():

```
if (array_key_exists(key, array)) { ... }
```

ترجع الدالة قيمة منطقية تشير إلى ما إذا كان المدخل الأول هو مفتاح صالح في المصفوفة المعطاة كمدخل ثاني.

لا يكفي أن تقول ببساطة:

```
if ($person['name']) { ... } // this can be misleading
```

حتى إذا كان هناك عنصر في المصفوفة يحمل اسم المفتاح، فقد تكون قيمته المقابلة خطأ (على سبيل المثال، 0 أو NULL أو السلسلة الفارغة) بدلاً من ذلك، استخدم ()array_key_exists، كما يلي:

```
$person['age'] = 0; // unborn?

if ($person['age']) {
 echo "true!\n";
}

if (array_key_exists('age', $person)) {
 echo "exists!\n";
}

exists!
```

يستخدم العديد من الأشخاص الدالة ()isset بدلاً من ذلك، والتي تُرجع true إذا كان العنصر موجودًا وليس NULL:

```
$a = array(0, NULL, '');

function tf($v)
{
 return $v ? 'T' : 'F';
}

for ($i=0; $i < 4; $i++) {
 printf("%d:    %s    %s\n",    $i,    tf(isset($a[$i])),
tf(array_key_exists($i, $a)));
}
0: T T
1: F T
2: T T
3: F F
```

## إزالة وإدراج العناصر في مصفوفة

يمكن للدالة ()array_splice إزالة العناصر أو إدراجها في مصفوفة وإنشاء مصفوفة أخرى اختياريًا من العناصر المُزالة:

```
$removed = array_splice(array, start [, length [,
replacement ] ]);
```

سنلقي نظرة على ()array_splice باستخدام هذه المصفوفة:

```
$subjects = array("physics", "chem", "math", "bio",
"cs", "drama", "classics");
```

يمكننا إزالة العناصر "math" و "bio" و "cs" بإخبار ()array_splice بالبدء من الموضع 2 وإزالة 3 عناصر:

```
$removed = array_splice($subjects, 2, 3);
// $removed is array("math", "bio", "cs")
// $subjects is array("physics", "chem", "drama",
"classics")
```

إذا حذفت الطول، فإن ()array_splice يزيل إلى نهاية المصفوفة:

```
$removed = array_splice($subjects, 2);
// $removed is array("math", "bio", "cs", "drama",
"classics")
// $subjects is array("physics", "chem")
```

إذا كنت تريد حذف العناصر من المصفوفة المصدر ولا تهتم بالاحتفاظ بقيمها، فلن تحتاج إلى تخزين نتائج ()array_splice:

```
array_splice($subjects, 2);
// $subjects is array("physics", "chem");
```

لإدراج العناصر حيث تمت إزالة العناصر الأخرى، استخدم المدخل الرابع:

```
$new = array("law", "business", "IS");
```

```
array_splice($subjects, 4, 3, $new);
// $subjects is array("physics", "chem", "math", "bio",
"law", "business", "IS")
```

لا يجب أن يكون حجم المصفوفة البديلة هو نفسه عدد العناصر التي تحذفها. تنمو المصفوفة أو تتقلص حسب الحاجة:

```
$new = array("law", "business", "IS");
array_splice($subjects, 3, 4, $new);
// $subjects is array("physics", "chem", "math", "law",
"business", "IS")
```

لإدراج عناصر جديدة في المصفوفة أثناء دفع العناصر الموجودة إلى اليمين، احذف العناصر الصفرية:

```
$subjects = array("physics", "chem", "math');
$new = array("law", "business");
array_splice($subjects, 2, 0, $new);
//  $subjects  is  array("physics",  "chem",  "law",
"business", "math")
```

على الرغم من أن الأمثلة حتى الآن استخدمت مصفوفة مفهرسة، فإن array_splice() يعمل أيضًا على المصفوفات الترابطية:

```
$capitals = array(
 'USA' => "Washington",
 'Great Britain' => "London",
 'New Zealand' => "Wellington",
```

```
 'Australia' => "Canberra",
 'Italy' => "Rome",
 'Canada' => "Ottawa"
);

$downUnder = array_splice($capitals, 2, 2); // remove New Zealand and Australia
$france = array('France' => "Paris");

array_splice($capitals, 1, 0, $france); // insert France between USA and GB
```

# التحويل بين المصفوفات والمتغيرات

توفر PHP دالتين، ()extract و ()compact، والتي تحول بين المصفوفات والمتغيرات. تتوافق أسماء المتغيرات مع المفاتيح الموجودة في المصفوفة، وتصبح قيم المتغيرات هي القيم في المصفوفة. على سبيل المثال، هذه المجموعة

```
$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Betty");
```

يمكن تحويلها أو بناؤها من هذه المتغيرات:

```
$name = "Fred";
$age = 35;
$wife = "Betty";
```

## تكوين متغيرات من مصفوفة

تقوم دالة ()extract تلقائيًا بإنشاء متغيرات محلية من مصفوفة. تصبح مؤشرات عناصر المصفوفة أسماء المتغيرات:

```
extract($person); // $name, $age, and $wife are now set
```

إذا كان المتغير الذي تم إنشاؤه بواسطة الاستخراج له نفس اسم متغير موجود، فسيتم استبدال قيمة المتغير الحالي بمتغير من المصفوفة.

يمكنك تعديل سلوك extract() عن طريق تمرير مدخل ثاني. يصف الملحق القيم المحتملة لهذا المدخل الثاني. القيمة الأكثر فائدة هي EXTR_PREFIX_ALL، مما يشير إلى أن المدخل الثالث لـ extract() هي بادئة لأسماء المتغيرات التي تم إنشاؤها. يساعد ذلك في ضمان إنشاء أسماء متغيرات فريدة عند استخدام extract(). من الجيد استخدام أسلوب PHP دائمًا باستخدام EXTR_PREFIX_ALL، كما هو موضح هنا:

```
$shape = "round";
$array = array('cover' => "bird", 'shape' =>
"rectangular");

extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: {$book_cover}, Book Shape: {$book_shape},
Shape: {$shape}";

Cover: bird, Book Shape: rectangular, Shape: round
```

## تكوين مصفوفة من المتغيرات

الدالة compact() هي عكس extract()، يمكنك تمرير أسماء المتغيرات لضغطها إما كمعلمات منفصلة أو في مصفوفة. تُنشئ الدالة compact() مصفوفة ترابطية تكون مفاتيحها أسماء المتغيرات وقيمها هي قيم المتغير. يتم تخطي أي أسماء في المصفوفة لا تتوافق مع المتغيرات الفعلية. في ما يلي مثال على عملية compact():

```
$color = "indigo";
$shape = "curvy";
$floppy = "none";

$a = compact("color", "shape", "floppy");
```

```
// or
$names = array("color", "shape", "floppy");
$a = compact($names);
```

# عبور المصفوفات

**Traversing Arrays**

تتمثل المهمة الأكثر شيوعاً في المصفوفات في القيام بشيء ما مع كل عنصر - على سبيل المثال، إرسال بريد إلى كل عنصر من عناصر مصفوفة من العناوين، أو تحديث كل ملف في مجموعة من أسماء الملفات، أو إضافة كل عنصر من عناصر مصفوفة الأسعار. هناك عدة طرق لاجتياز المصفوفات في PHP، وستعتمد الطريقة التي تختارها على بياناتك والمهمة التي تقوم بها.

### بناء foreach

الطريقة الأكثر شيوعاً للتكرار حول عناصر المصفوفة هي استخدام بنية foreach:

```
$addresses = array("spam@cyberpromo.net",
"abuse@example.com");

foreach ($addresses as $value) {
 echo "Processing {$value}\n";
}
Processing spam@cyberpromo.net
Processing abuse@example.com
```

ينفذ PHP جسم الحلقة (جملة echo) مرة واحدة لكل عنصر من عناصر $addresses بدوره، مع تعيين $value على العنصر الحالي. تتم معالجة العناصر بترتيبها الداخلي.

يتيح لك شكل بديل من foreach الوصول إلى المفتاح الحالي:

```
$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Wilma");

foreach ($person as $key => $value) {
 echo "Fred's {$key} is {$value}\n";
}
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

في هذه الحالة، يتم وضع مفتاح كل عنصر في $key ويتم وضع القيمة المقابلة في $value.

لا تعمل بنية foreach على المصفوفة نفسها، بل تعمل على نسخة منها. يمكنك إدراج العناصر أو حذفها في نص حلقة foreach، مع العلم أن الحلقة لن تحاول معالجة العناصر المحذوفة أو المدرجة.

## دوال التكرار

كل مصفوفة PHP تتعقب العنصر الحالي الذي تعمل به؛ يُعرف المؤشر إلى العنصر الحالي بالمكرر "*iterator*". PHP لديها دوال لتعيين هذا المكرر ونقله وإعادة تعيينه. دوال المكرر هي:

***current()***

إرجاع العنصر المشار إليه حالياً بواسطة المكرر.

***reset()***

ينقل المكرر إلى العنصر الأول في المصفوفة ويعيده.

***next()***

ينقل المكرر إلى العنصر التالي في المصفوفة ويعيده.

***prev()***

ينقل المكرر إلى العنصر السابق في المصفوفة ويعيده.

***end()***

ينقل المكرر إلى آخر عنصر في المصفوفة ويعيده.

***each()***

تُرجع المفتاح والقيمة للعنصر الحالي كمصفوفة وتحريك المكرر إلى العنصر التالي في المصفوفة.

***key()***

إرجاع مفتاح العنصر الحالي.

تُستخدم دالة each() لتكرار عناصر المصفوفة. يعالج العناصر وفقًا لترتيبها الداخلي:

```
reset($addresses);

while (list($key, $value) = each($addresses)) {
 echo "{$key} is {$value}<br />\n";
}
0 is spam@cyberpromo.net
```

```
1 is abuse@example.com
```

هذا النهج لا يصنع نسخة من المصفوفة، كما يفعل foreach. هذا مفيد للمصفوفات الكبيرة جدًا عندما تريد توفير الذاكرة.

تكون دوال المكرر مفيدة عندما تحتاج إلى النظر في بعض أجزاء المصفوفة بشكل منفصل عن الأجزاء الأخرى. يوضح المثال 5-1 التعليمات البرمجية التي تنشئ جدولاً، وتعامل الفهرس الأول والقيمة في مصفوفة ترابطية كعناوين أعمدة في الجدول.

مثال 5-1. بناء جدول بوظائف التكرار

```
$ages = array(
 'Person' => "Age",
 'Fred' => 35,
 'Barney' => 30,
 'Tigger' => 8,
 'Pooh' => 40
);

// start table and print heading
reset($ages);

list($c1, $c2) = each($ages);
```

```
echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n");

// print the rest of the values
while (list($c1, $c2) = each($ages)) {
 echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// end the table
echo("</table>");
```

### باستخدام حلقة for

إذا كنت تعلم أنك تتعامل مع مصفوفة مفهرسة، حيث المفاتيح عبارة عن أعداد صحيحة متتالية تبدأ من 0، فيمكنك استخدام حلقة for للعد من خلال المؤشرات. تعمل الحلقة for على المصفوفة نفسها، وليس على نسخة من المصفوفة، وتعالج العناصر بترتيب المفاتيح بغض النظر عن ترتيبها الداخلي.

إليك كيفية طباعة مصفوفة باستخدام for:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");
$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
 $value = $addresses[$i];
 echo "{$value}\n";
}
```

spam@cyberpromo.net

abuse@example.com

### استدعاء دالة لكل عنصر مصفوفة

توفر PHP آلية، ()array_walk، لاستدعاء دالة معرفة من قبل المستخدم مرة واحدة لكل عنصر في مصفوفة:

```
array_walk(array, callable);
```

تأخذ الدالة التي تحددها مدخلين أو ثلاثة اختياريًا: الأولى هي قيمة العنصر، والثانية هي مفتاح العنصر، والثالثة هي القيمة المقدمة إلى ()array_walk عند استدعائها. على سبيل المثال، إليك طريقة أخرى لطباعة أعمدة جدول مكونة من قيم من مصفوفة:

```
$printRow = function ($value, $key)
{
 print("<tr><td>{$key}</td><td>{$value}</td></tr>\n");
};

$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Wilma");

echo "<table border=1>";

array_walk($person, $printRow);

echo "</table>";
```

يحدد أحد أشكال هذا المثال لون الخلفية باستخدام المدخل الثالث الاختياري لـ ()array_walk، تمنحنا هذه المعلمة المرونة التي نحتاجها لطباعة العديد من الجداول، مع العديد من ألوان الخلفية:

```
function printRow($value, $key, $color)
{
 echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
 echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n";
}

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

echo "<table border=\"1\">";

array_walk($person, "printRow", "lightblue");
echo "</table>";
```

إذا كانت لديك خيارات متعددة تريد تمريرها إلى الدالة التي تم استدعاؤها، فما عليك سوى تمرير مصفوفة كمعلمة ثالثة:

```
$extraData = array('border' => 2, 'color' => "red");
$baseArray = array("Ford", "Chrysler", "Volkswagen", "Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);
```

```
function walkFunction($item, $index, $data)
{
 echo "{$item} <- item, then border: {$data['border']}";
 echo " color->{$data['color']}<br />" ;
}
Ford <- item, then border: 2 color->red
Crysler <- item, then border: 2 color->red
VW <- item, then border: 2 color->red
Honda <- item, then border: 2 color->red
Toyota <- item, then border: 2 color->red
```

تعالج الدالة ()array_walk العناصر بترتيبها الداخلي.

## اختزال المصفوفة Reducing an Array

ابن عم ()array_walk، ()array_reduce يطبق دالة على كل عنصر من المصفوفة بدوره، لبناء قيمة واحدة:

```
$result = array_reduce(array, callable [, default ]);
```

تأخذ الدالة مدخلين: الإجمالي الجاري "running total"، والقيمة الحالية "current value" قيد المعالجة. يجب أن يعيد إجمالي التشغيل الجديد. على سبيل المثال، لإضافة مربعات قيم المصفوفة، استخدم:

```
$addItUp = function ($runningTotal, $currentValue)
{
 $runningTotal += $currentValue * $currentValue;
```

```
 return $runningTotal;
};

$numbers = array(2, 3, 5, 7);
$total = array_reduce($numbers, $addItUp);

echo $total;
```

```
87
```

يقوم سطر ()array_reduce باستدعاء هذه الدالة:

```
addItUp(0, 2);
addItUp(4, 3);
addItUp(13, 5);
addItUp(38, 7);
```

المدخل الافتراضي، إذا تم توفيره، هو قيمة أولية. على سبيل المثال، إذا قمنا بتغيير الاستدعاء إلى ()array_reduce في المثال السابق إلى:

```
$total = array_reduce($numbers, "addItUp", 11);
```

استدعاءات الدالة الناتجة هي:

```
addItUp(11, 2);
addItUp(15, 3);
```

```
addItUp(24, 5);
addItUp(49, 7);
```

إذا كانت المصفوفة فارغة، تُرجع الدالة array_reduce() القيمة الافتراضية "*default*". إذا لم يتم إعطاء قيمة افتراضية وكانت المصفوفة فارغة، فإن array_reduce() ترجع NULL.

## البحث عن القيم

ترجع الدالة in_array() صح "true" أو خطأً "false"، اعتمادًا على ما إذا كان المدخل الأول عنصرًا في المصفوفة باعتبارها المدخل الثاني:

```
if (in_array(to_find, array [, strict])) { ... }
```

إذا كان المدخل الثالث الاختياري صحيح "true"، فيجب أن تتطابق أنواع to_find والقيمة في المصفوفة. الافتراضي هو عدم التحقق من أنواع البيانات.

إليك مثال بسيط:

```
$addresses = array("spam@cyberpromo.net",
"abuse@example.com",
"root@example.com");
$gotSpam = in_array("spam@cyberpromo.net", $addresses);
// $gotSpam is true
```

```
$gotMilk = in_array("milk@tucows.com", $addresses); //
$gotMilk is false
```

تقوم PHP تلقائيًا بفهرسة القيم في المصفوفات، لذا فإن in_array() بشكل عام أسرع بكثير من قيام حلقة بفحص كل قيمة في المصفوفة للعثور على القيمة التي تريدها.

يتحقق المثال 5-2 مما إذا كان المستخدم قد أدخل المعلومات في جميع الحقول المطلوبة في النموذج.

مثال 5-2. البحث في المصفوفة

```
<?php
function hasRequired($array, $requiredFields) {
 $array =

 $keys = array_keys ( $array );
 foreach ( $requiredFields as $fieldName ) {
 if (! in_array ( $fieldName, $keys )) {
 return false;
 }
 }
 return true;
}
if ($_POST ['submitted']) {
 $testArray = array_filter($_POST);
 echo "<p>You ";
```

```
echo hasRequired ( $testArray, array (
'name',
'email_address'
) ) ? "did" : "did not";
echo " have all the required fields.</p>";
}
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p>
Name: <input type="text" name="name" /><br /> Email address: <input
type="text" name="email_address" /><br /> Age (optional): <input
type="text" name="age" />
</p>
<p align="center">
<input type="submit" value="submit" name="submitted" />
</p>
</form>
```

الاختلاف في ()in_array هو دالة ()array_search. أثناء إرجاع ()in_array صحيحاً إذا تم العثور على القيمة، تقوم ()array_search بإرجاع مفتاح العنصر، إذا وجد:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$k = array_search("Wilma", $person);
```

```
echo("Fred's {$k} is Wilma\n");
```

```
Fred's wife is Wilma
```

تأخذ الدالة ()array_search أيضًا المدخل الصارم "*strict*" الثالث الاختياري، والتي تتطلب تطابق أنواع القيمة التي يتم البحث عنها والقيمة الموجودة في المصفوفة.

# فرز Sorting

يغير الفرز الترتيب الداخلي للعناصر في المصفوفة ويعيد كتابة المفاتيح اختياريًا لتعكس هذا الترتيب الجديد. على سبيل المثال، قد تستخدم الفرز لترتيب قائمة بالدرجات من الأكبر إلى الأصغر، أو لترتيب قائمة الأسماء أبجديًا، أو لترتيب مجموعة من المستخدمين بناءً على عدد الرسائل التي قاموا بنشرها.

توفر PHP ثلاث طرق لفرز المصفوفات - الفرز حسب المفاتيح، والفرز حسب القيم دون تغيير المفاتيح، أو الفرز حسب القيم ثم تغيير المفاتيح. يمكن إجراء كل نوع من أنواع الفرز بترتيب تصاعدي أو تنازلي أو ترتيب تحدده دالة محددة بواسطة المستخدم.

## فرز مصفوفة واحدة في كل مرة

تظهر الدوال التي توفرها PHP لفرز مصفوفة في الجدول 5-1.

الجدول 5-1. دوال PHP لفرز المصفوفة

| **User-defined order** ترتيب محدد من قبل المستخدم | **Descending** تنازلي | **Ascending** تصاعدي | التأثير |
|---|---|---|---|
| usort() | rsort() | sort() | افرز المصفوفة بالقيم، ثم إعد تعيين الفهارس بدءا من 0 |
| uasort() | arsort() | asort() | فرز المصفوفة حسب القيم |
| uksort() | krsort() | ksort() | فرز المصفوفة حسب المفاتيح |

تم تصميم دوال sort() و rsort() و usort() للعمل على المصفوفات المفهرسة لأنها تعين مفاتيح رقمية جديدة لتمثيل الترتيب. إنها مفيدة عندما تحتاج إلى الإجابة عن أسئلة مثل "ما هي أعلى 10 نقاط؟" و "من هو الشخص الثالث حسب الترتيب الأبجدي؟" يمكن استخدام دوال الفرز الأخرى في المصفوفات المفهرسة، لكنك ستتمكن فقط من الوصول إلى الترتيب الذي تم فرزه باستخدام تراكيب الاجتياز مثل foreach و next().

لفرز الأسماء بترتيب أبجدي تصاعدي، افعل شيئًا كالتالي:

```
$names = array("Cath", "Angela", "Brad", "Mira");
sort($names); // $names is now "Angela", "Brad", "Cath",
"Mira"
```

للحصول عليها بترتيب أبجدي عكسي، ما عليك سوى استدعاء ()rsort بدلاً من ()sort.

إذا كان لديك مصفوفة ترابطية تقوم بتعيين أسماء المستخدمين إلى دقائق من وقت تسجيل الدخول، يمكنك استخدام ()arsort لعرض جدول من الثلاثة الأوائل، كما هو موضح هنا:

```
$logins = array(
 'njt' => 415,
 'kt' => 492,
 'rl' => 652,
 'jht' => 441,
 'jj' => 441,
 'wt' => 402,
 'hut' => 309,
);

arsort($logins);

$numPrinted = 0;

echo "<table>\n";

foreach ($logins as $user => $time) {
 echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");

 if (++$numPrinted == 3) {
 break; // stop after three
```

```
 }
}

echo "</table>";
```

إذا كنت تريد عرض الجدول بترتيب تصاعدي حسب اسم المستخدم، فاستخدم ()ksort بدلاً من ذلك.

يتطلب الترتيب المعرف من قبل المستخدم توفير دالة تأخذ قيمتين وترجع قيمة تحدد ترتيب القيمتين في المصفوفة التي تم فرزها. يجب أن ترجع الدالة 1 إذا كانت القيمة الأولى أكبر من الثانية، و 1- إذا كانت القيمة الأولى أقل من الثانية، و 0 إذا كانت القيم هي نفسها لأغراض ترتيب الفرز المخصص.

يقوم البرنامج في المثال 5-3 بتطبيق دوال الفرز المختلفة على نفس البيانات.

المثال 5-3. فرز المصفوفات

```
<?php
function userSort($a, $b)
{
 // smarts is all-important, so sort it first
 if ($b == "smarts") {
 return 1;
 }
 else if ($a == "smarts") {
```

```
 return -1;
 }

 return ($a == $b) ? 0 : (($a < $b) ? -1 : 1);
}

$values = array(
 'name' => "Buzz Lightyear",
 'email_address' => "buzz@starcommand.gal",
 'age' => 32,
 'smarts' => "some"
);

if ($_POST['submitted']) {
 $sortType = $_POST['sort_type'];

 if ($sortType == "usort" || $sortType == "uksort" ||
$sortType == "uasort") {
 $sortType($values, "userSort");
 }
 else {
 $sortType($values);
 }
} ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?> "
method="post">
 <p>
```

```
<input type="radio" name="sort_type"
value="sort" checked="checked" /> Standard<br />
<input type="radio" name="sort_type" value="rsort" />
Reverse<br />
<input type="radio" name="sort_type" value="usort" />
User-defined<br />
<input type="radio" name="sort_type" value="ksort" />
Key<br />
<input type="radio" name="sort_type" value="krsort" />
Reverse key<br />
<input type="radio" name="sort_type"
value="uksort" /> User-defined key<br />
<input type="radio" name="sort_type" value="asort" />
Value<br />
<input type="radio" name="sort_type"
value="arsort" /> Reverse value<br />
<input type="radio" name="sort_type"
value="uasort" /> User-defined value<br />
</p>

<p align="center"><input type="submit" value="Sort"
name="submitted" /></p>

<p>Values <?php echo $_POST['submitted'] ? "sorted by
{$sortType}" : "unsorted";
?>:</p>

<ul>
```

```
 <?php foreach ($values as $key => $value) {
 echo "<li><b>{$key}</b>: {$value}</li>";
 } ?>
 </ul>
</form>
```

## فرز الترتيب الطبيعي

دوال الفرز المضمنة في PHP تقوم بفرز السلاسل والأرقام بشكل صحيح، لكنها لا تقوم بفرز السلاسل التي تحتوي على أرقام بشكل صحيح. على سبيل المثال، إذا كان لديك أسماء الملفات ex10.php و ex5.php و ex1.php، فإن دوال الفرز العادية ستعيد ترتيبها بالترتيب التالي: ex1.php، ex10.php، ex5.php. لفرز السلاسل التي تحتوي على أرقام بشكل صحيح، استخدم دالتي natsort() و natcasesort():

```
$output = natsort(input);
$output = natcasesort(input);
```

## فرز المصفوفات المتعددة مرة واحدة

تقوم الدالة array_multisort() بفرز المصفوفات المفهرسة المتعددة مرة واحدة:

```
array_multisort(array1 [, array2, ... ]);
```

قم بتمرير سلسلة من المصفوفات وأوامر الفرز (المحددة بواسطة ثوابت SORT_ASC أو SORT_DESC)، وتقوم بإعادة ترتيب عناصر جميع المصفوفات، وتعيين فهارس جديدة. إنه مشابه لعملية الانضمام join على قاعدة بيانات علائقية.

تخيل أن لديك الكثير من الأشخاص وعدة أجزاء من البيانات عن كل شخص:

```
$names = array("Tom", "Dick", "Harriet", "Brenda",
"Joe");
$ages = array(25, 35, 29, 35, 35);
$zips = array(80522, '02140', 90210, 64141, 80522);
```

يمثل العنصر الأول من كل مصفوفة سجلاً منفردًا — كل المعلومات المعروفة عن Tom. وبالمثل، فإن العنصر الثاني يشكل تسجيلة أخرى - كل المعلومات المعروفة عن Dick. تعيد الدالة array_multisort() ترتيب عناصر المصفوفات، مع الاحتفاظ بالسجلات. أي، إذا انتهى الأمر بـ "Dick" أولاً في مصفوفة names$بعد الفرز، فستكون بقية معلومات Dick أولاً في المصفوفات الأخرى أيضًا. (لاحظ أننا احتجنا إلى اقتباس الرمز البريدي الخاص بـ Dick لمنع تفسيره على أنه ثابت ثماني).

إليك كيفية فرز السجلات أولاً تصاعديًا حسب العمر ، ثم تنازليًا حسب الرمز البريدي:

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC,
$names, SORT_ASC);
```

نحتاج إلى تضمين names$ في استدعاء الدالة للتأكد من بقاء اسم Dick مع عمره والرمز البريدي. تظهر طباعة البيانات نتيجة الفرز:

```
for ($i = 0; $i < count($names); $i++) {
```

```
 echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
Tom, 25, 80522
Harriet, 29, 90210
Joe, 35, 80522
Brenda, 35, 64141
Dick, 35, 02140
```

## عكس المصفوفات

تعكس الدالة ()array_reverse الترتيب الداخلي للعناصر في المصفوفة:

```
$reversed = array_reverse(array);
```

يتم إعادة ترقيم المفاتيح الرقمية بدءًا من 0، بينما لا تتأثر فهارس السلسلة. بشكل عام، من الأفضل استخدام دوال الترتيب العكسي بدلاً من الفرز ثم عكس ترتيب المصفوفة.

تُرجع الدالة ()array_flip مصفوفة تعكس ترتيب زوج القيمة والمفتاح لكل عنصر أصلي:

```
$flipped = array_flip(array);
```

أي، لكل عنصر في المصفوفة قيمته مفتاح صالح، تصبح قيمة العنصر مفتاحه ويصبح مفتاح العنصر قيمته. على سبيل المثال، إذا كان لديك مصفوفة تقوم بتعيين أسماء المستخدمين إلى مجلدات الصفحة الرئيسية، فيمكنك استخدام ()array_flip لإنشاء مصفوفة تقوم بتعيين المجلدات الرئيسية لأسماء المستخدمين:

```
$u2h = array(
 'gnat' => "/home/staff/nathan",
 'frank' => "/home/action/frank",
```

```
 'petermac' => "/home/staff/petermac",
 'ktatroe' => "/home/staff/kevin"
);
$h2u = array_flip($u2h);

$user = $h2u["/home/staff/kevin"]; // $user is now 'ktatroe'
```

العناصر التي لا تكون قيمها الأصلية سلاسل أو أعداد صحيحة تُترك بمفردها في المصفوفة الناتجة. تتيح لك المصفوفة الجديدة اكتشاف المفتاح في المصفوفة الأصلية نظراً لقيمتها، ولكن هذه التقنية تعمل بفعالية فقط عندما يكون للمصفوفة الأصلية قيم فريدة.

## ترتيب العشوائية

لاجتياز العناصر في مصفوفة بترتيب عشوائي، استخدم الدالة ()shuffle. يستبدل جميع المفاتيح الموجودة - سلسلة أو رقمية - بأعداد صحيحة متتالية تبدأ من 0.

إليك كيفية ترتيب أيام الأسبوع عشوائيًا:

```
$weekdays = array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday");
shuffle($weekdays);

print_r($weekdays);

Array(
```

```
 [0] => Tuesday
 [1] => Thursday
 [2] => Monday
 [3] => Friday
 [4] => Wednesday
)
```

من الواضح أن الترتيب بعد shuffle() قد لا يكون هو نفسه إخراج العينة هنا نظرًا للطبيعة العشوائية للدالة. ما لم تكن مهتمًا بالحصول على عدة عناصر عشوائية من مصفوفة دون تكرار أي عنصر محدد، فإن استخدام الدالة rand() لاختيار فهرس يكون أكثر كفاءة.

# العمل على المصفوفات بأكملها

تحتوي PHP على العديد من الدوال المضمنة المفيدة لتعديل أو تطبيق عملية على جميع عناصر المصفوفة. يمكنك حساب مجموع مصفوفة، ودمج عدة مصفوفات، وإيجاد الفرق بين مصفوفتين، والمزيد.

## حساب مجموع المصفوفة

تجمع الدالة ()array_sum القيم في مصفوفة مفهرسة أو ترابطية:

```
$sum = array_sum(array);
```

كمثال:

```
$scores = array(98, 76, 56, 80);
$total = array_sum($scores); // $total = 310
```

دمج مصفوفتين

تدمج الدالة ()array_merge بذكاء مصفوفتين أو أكثر:

```
$merged = array_merge(array1, array2 [, array ... ])
```

إذا تم تكرار مفتاح رقمي من مصفوفة سابقة، فسيتم تعيين مفتاح رقمي جديد للقيمة من المصفوفة اللاحقة:

```
$first = array("hello", "world"); // 0 => "hello", 1 =>
"world"
```

```
$second = array("exit", "here"); // 0 => "exit", 1 =>
"here"


$merged = array_merge($first, $second);
// $merged = array("hello", "world", "exit", "here")
```

إذا تم تكرار مفتاح سلسلة من مصفوفة سابقة، فسيتم استبدال القيمة السابقة بالقيمة اللاحقة:

```
$first = array('bill' => "clinton", 'tony' => "danza");
$second = array('bill' => "gates", 'adam' => "west");


$merged = array_merge($first, $second);
// $merged = array('bill' => "gates", 'tony' => "danza",
'adam' => "west")
```

حساب الفرق بين مصفوفتين

تحسب الدالة ()array_diff الفرق بين مصفوفتين أو أكثر، وتعيد مصفوفة بقيم من المصفوفة الأولى غير موجودة في المصفوفات الأخرى:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

كمثال:

```
$a1 = array("bill", "claire", "ella", "simon", "judy");
$a2 = array("jack", "claire", "toni");
$a3 = array("ella", "simon", "garfunkel");
```

```
// find values of $a1 not in $a2 or $a3
$difference = array_diff($a1, $a2, $a3);
print_r($difference);

Array(
 [0] => "bill",
 [4] => "judy"
);
```

تتم مقارنة القيم باستخدام عامل المقارنة الصارم ===، لذلك يتم اعتبار 1 و "1" مختلفين. يتم الاحتفاظ بمفاتيح المصفوفة الأولى، لذا في $diff يكون مفتاح "bill" هو 0 ومفتاح "judy" هو 4.

في مثال آخر، يُرجع الكود التالي الفرق بين مصفوفتين:

```
$first = array(1, "two", 3);
$second = array("two", "three", "four");

$difference = array_diff($first, $second);
print_r($difference);

Array(
 [0] => 1
 [2] => 3
)
```

## تصفية العناصر من مصفوفة

لتحديد مجموعة فرعية من مصفوفة بناءً على قيمها، استخدم الدالة ()array_filter:

```
$filtered = array_filter(array, callback);
```

يتم تمرير كل قيمة مصفوفة *"array"* إلى الدالة المسماة في رد الاتصال *"callback"*. تحتوي المصفوفة التي تم إرجاعها فقط على عناصر المصفوفة الأصلية التي ترجع لها الدالة قيمة صحيحة. فمثلا:

```
function isOdd ($element) {
 return $element % 2;
}

$numbers = array(9, 23, 24, 27);
$odds = array_filter($numbers, "isOdd");

// $odds is array(0 => 9, 1 => 23, 3 => 27)
```

كما ترى، يتم الاحتفاظ بالمفاتيح. هذه الدالة مفيدة للغاية مع المصفوفات الترابطية.

# استخدام المصفوفات لتنفيذ أنواع البيانات

**Using Arrays to Implement Data Types**

تظهر المصفوفات في كل برنامج PHP تقريبًا. بالإضافة إلى الغرض الواضح من تخزين مجموعات القيم، يتم استخدامها أيضًا لتنفيذ أنواع بيانات مجردة متنوعة. في هذا القسم، نوضح كيفية استخدام المصفوفات لتنفيذ المجموعات "sets" والمكدسات "stacks".

## مجموعات

تمكّنك المصفوفات من تنفيذ العمليات الأساسية لنظرية المجموعات: الاتحاد "union" والتقاطع "intersection" والاختلاف "difference". يتم تمثيل كل مجموعة بمصفوفة، وتقوم دوال PHP المختلفة بتنفيذ عمليات المجموعة. القيم الموجودة في المجموعة هي القيم الموجودة في المصفوفة — لا يتم استخدام المفاتيح، ولكن يتم الاحتفاظ بها بشكل عام بواسطة العمليات.

اتحاد مجموعتين "*union*" هو كل العناصر من كلا المجموعتين مع إزالة التكرارات. تتيح لك الدالتان array_merge() و array_unique() حساب الاتحاد. إليك كيفية العثور على اتحاد مصفوفتين:

```
function arrayUnion($a, $b)
{
 $union = array_merge($a, $b); // duplicates may still
exist
 $union = array_unique($union);

 return $union;
```

```
}


$first = array(1, "two", 3);
$second = array("two", "three", "four");


$union = arrayUnion($first, $second);
print_r($union);


Array(
 [0] => 1
 [1] => two
 [2] => 3
 [4] => three
 [5] => four
)
```

تقاطع *"intersection"* مجموعتين هو مجموعة العناصر المشتركة بينهما. تأخذ دالة array_intersect() المضمنة في PHP أي عدد من المصفوفات كمدخلات وتعيد مصفوفة من تلك القيم الموجودة في كل منها. إذا كانت المفاتيح المتعددة لها نفس القيمة، فسيتم الاحتفاظ بالمفتاح الأول بهذه القيمة.

## المكدسات "Stacks"

على الرغم من أنه ليس شائعًا في برامج PHP كما هو الحال في البرامج الأخرى، إلا أن أحد أنواع البيانات الشائعة إلى حد ما هو آخر مكدس يدخل أولاً "last-in first-out " (LIFO). يمكننا إنشاء حزم باستخدام زوج من دوال PHP، array_push() و array_pop(). الدالة array_push() مطابقة

لإسناد []$array. نستخدم ()array_push لأنه يبرز حقيقة أننا نعمل مع الحزم "stacks"، والتوازي مع ()array_pop يجعل قراءة الكود أسهل. هناك أيضًا دالات ()array_shift و ()array_unshift لمعاملة مصفوفة مثل قائمة انتظار.

الأكوام "Stacks" مفيدة بشكل خاص للحفاظ على الحالة. يوفر المثال 5-4 مصححًا بسيطًا للحالة يسمح لك بطباعة قائمة بالدوال التي تم استدعاؤها حتى هذه النقطة (أي تتبع المكدس).

مثال 5-4. مصحح أخطاء الحالة

```
$callTrace = array();

function enterFunction($name)
{
 global $callTrace;
 $callTrace[] = $name;

 echo "Entering {$name} (stack is now: " . join(' -> ',
$callTrace) . ")<br />";
}

function exitFunction()
{
 echo "Exiting<br />";

 global $callTrace;
```

```
 array_pop($callTrace);
}

function first()
{
 enterFunction("first");
 exitFunction();
}

function second()
{
 enterFunction("second");
 first();
 exitFunction();
}

function third()
{
 enterFunction("third");
 second();
 first();
 exitFunction();
}

first();
third();
```

هذا هو الناتج من المثال 5-4:

```
Entering first (stack is now: first)
Exiting
Entering third (stack is now: third)
Entering second (stack is now: third -> second)
Entering first (stack is now: third -> second -> first)
Exiting
Exiting
Entering first (stack is now: third -> first)
Exiting
Exiting
```

# تنفيذ واجهة التكرار

**Implementing the Iterator Interface**

باستخدام بنية foreach، يمكنك التكرار ليس فقط عبر المصفوفات، ولكن أيضًا عبر حالات الفئات "classes" التي تنفذ واجهة Iterator (انظر الفصل 6 لمزيد من المعلومات حول الكائنات والواجهات). لتنفيذ واجهة المكرر Iterator، يجب عليك تنفيذ خمس طرق في فئتك:

***current()***

إرجاع العنصر المشار إليه حاليًا بواسطة المكرر.

***key()***

إرجاع مفتاح العنصر المشار إليه حاليًا بواسطة المكرر.

***next()***

ينقل المكرر إلى العنصر التالي في الكائن ويعيده.

***rewind()***

ينقل المكرر إلى العنصر الأول في المصفوفة.

***valid()***

يرجع true إذا كان المكرر يشير حاليًا إلى عنصر صالح، ويعيد النتيجة false إذا كان العكس.

مثال 5-5 يعيد تطبيق فئة مكرر بسيطة تحتوي على مصفوفة ثابتة من البيانات.

مثال 5-5. واجهة التكرار

```
class BasicArray implements Iterator
{
 private $position = 0;
 private $array = ["first", "second", "third"];

 public function __construct()
 {
 $this->position = 0;
 }

 public function rewind()
 {
 $this->position = 0;
 }

 public function current()
 {
 return $this->array[$this->position];
 }

 public function key()
 {
 return $this->position;
 }
```

```
 public function next()
 {
 $this->position += 1;
 }

 public function valid()
 {
 return isset($this->array[$this->position]);
 }
}

$basicArray = new BasicArray;

foreach ($basicArray as $value) {
 echo "{$value}\n";
}

foreach ($basicArray as $key => $value) {
 echo "{$key} => {$value}\n";
}

first
second
third

0 => first
```

```
1 => second
2 => third
```

عندما تقوم بتطبيق واجهة Iterator على فئة، فإنها تسمح لك فقط باجتياز العناصر في حالات تلك الفئة باستخدام بنية foreach؛ لا يسمح لك بمعاملة تلك الحالات كمصفوفات أو معلمات لطرق أخرى. هذا، على سبيل المثال، يعيد التكرار الذي يشير إلى خصائص $trie باستخدام دالة rewind() المضمنة بدلاً من استدعاء طريقة rewind() على $trie:

```
class Trie implements Iterator
{
 const POSITION_LEFT = "left";
 const POSITION_THIS = "this";
 const POSITION_RIGHT = "right";

 var $leftNode;
 var $rightNode;

 var $position;

 // implement Iterator methods here...
}

$trie = new Trie();

rewind($trie);
```

توفر مكتبة SPL الاختيارية مجموعة متنوعة من التكرارات المفيدة، بما في ذلك دليل نظام الملفات "filesystem directory"، والشجرة "tree"، ومكررات مطابقة regex.

# مالتالي

غطت الفصول الثلاثة الأخيرة - حول الدوال، والسلاسل، والمصفوفات - الكثير من الأرضية التأسيسية. يبني الفصل التالي على هذا الأساس ويأخذك إلى عالم جديد من الكائنات والبرمجة الموجهة للكائنات "object-oriented programming" (OOP). يجادل البعض بأن OOP هي أفضل طريقة للبرمجة، لأنها مغلفة وقابلة لإعادة الاستخدام أكثر من البرمجة الإجرائية "procedural". يستمر هذا النقاش، ولكن بمجرد الدخول في النهج الموجه للكائنات في البرمجة وفهم فوائدها، يمكنك اتخاذ قرار مستنير حول كيفية البرمجة في المستقبل. ومع ذلك، فإن الاتجاه العام في عالم البرمجة هو استخدام OOP قدر الإمكان.

كلمة تحذير واحدة قبل المتابعة: هناك العديد من المواقف التي يمكن أن يضيع فيها مبرمج OOP المبتدئ، لذا تأكد من أنك مرتاح حقًا لـ OOP قبل القيام بأي شيء رئيسي أو مهمة حرجة معها.

# الفصل السادس: الكائنات

**Objects**

ستتعلم في هذا الفصل كيفية تحديد وإنشاء واستخدام الكائنات في PHP. تفتح البرمجة الموجهة للكائنات "Object-oriented programming" (OOP) الباب أمام تصميمات أنظف وصيانة أسهل وإعادة استخدام أكواد أكبر. أثبتت OOP أنها ذات قيمة كبيرة لدرجة أن القليل اليوم يجرؤ على تقديم لغة لم تكن موجهة نحو الهدف. يدعم PHP العديد من الميزات المفيدة لـ OOP، ويوضح لك هذا الفصل كيفية استخدامها، ويغطي مفاهيم OOP الأساسية بالإضافة إلى الموضوعات المتقدمة مثل الاستبطان "introspection" والتسلسل "serialization".

# كائنات Objects

تقر البرمجة الموجهة للكائنات بالاتصال الأساسي بين البيانات والكود الذي يعمل عليها، وتتيح لك تصميم وتنفيذ البرامج حول هذا الاتصال. على سبيل المثال، عادةً ما يتتبع نظام لوحة الإعلانات العديد من المستخدمين. في لغة البرمجة الإجرائية، يتم تمثيل كل مستخدم بهيكل بيانات، ومن المحتمل أن تكون هناك مجموعة من الدوال التي تعمل مع هياكل البيانات هذه (لإنشاء مستخدمين جدد، والحصول على معلوماتهم، وما إلى ذلك). في لغة OOP، يتم تمثيل كل مستخدم بواسطة كائن - بنية بيانات مع كود مرفق. لا تزال البيانات والكود موجودًا، ولكن يتم التعامل معها كوحدة لا يمكن فصلها. الكائن، باعتباره اتحادًا للكود والبيانات، هو الوحدة النمطية لتطوير التطبيقات وإعادة استخدام الكود.

في هذا التصميم الافتراضي للوحة الإعلانات، يمكن أن تمثل الكائنات ليس فقط المستخدمين ولكن أيضًا الرسائل "messages" و السلاسل "threads". كائن المستخدم له اسم مستخدم وكلمة مرور لهذا المستخدم، وكود لتعريف جميع الرسائل بواسطة هذا المؤلف. يعرف كائن الرسالة الخيط "thread" الذي ينتمي إليه ولديه كود لنشر رسالة جديدة والرد على رسالة موجودة وعرض الرسائل. كائن مؤشر الترابط هو مجموعة من كائنات الرسائل، وله كود لعرض فهرس الموضوع. هذه طريقة واحدة فقط لتقسيم الدوال الضرورية إلى كائنات. على سبيل المثال، في تصميم بديل، توجد التعليمات البرمجية لنشر رسالة جديدة في كائن المستخدم، وليس كائن الرسالة.

يعد تصميم الأنظمة الموجهة للكائنات موضوعًا معقدًا، وقد تمت كتابة العديد من الكتب حوله. الخبر السار هو أنه مهما كان تصميمك لنظامك، يمكنك تنفيذه في PHP. لنبدأ بتقديم بعض المصطلحات والمفاهيم الأساسية التي ستحتاج إلى معرفتها قبل الغوص في منهج البرمجة هذا.

# المصطلح Terminology

يبدو أن كل لغة موجهة للكائنات لديها مجموعة مختلفة من المصطلحات لنفس المفاهيم القديمة. يصف هذا القسم المصطلحات التي تستخدمها PHP، ولكن احذر من أنه في اللغات الأخرى قد يكون لهذه المصطلحات معاني أخرى.

لنعد إلى مثال مستخدمي لوحة الإعلانات. تحتاج إلى تتبع نفس المعلومات لكل مستخدم، ويمكن استدعاء نفس الدوال في بنية بيانات كل مستخدم. عندما تقوم بتصميم البرنامج، فإنك تقرر الحقول لكل مستخدم وتخرج بالدوال. بجمل OOP، أنت تصمم فئة المستخدم. الفئة عبارة عن قالب لبناء الكائنات.

**الكائن** "*object*" هو مثيل "instance" (أو حدوث "occurrence") لفئة. في هذه الحالة، إنها بنية بيانات مستخدم فعلية مع كود مرفق. الكائنات والفئات تشبه إلى حد ما القيم وأنواع البيانات. يوجد نوع بيانات واحد فقط لعدد صحيح، ولكن هناك العديد من الأعداد الصحيحة الممكنة. وبالمثل، يحدد برنامجك فئة مستخدم واحدة فقط، ولكن يمكنه إنشاء العديد من المستخدمين المختلفين (أو المتطابقين) منه.

تسمى البيانات المرتبطة بالكائن بخصائصه "*properties*". تسمى الدوال المرتبطة بالكائن بأساليبه "*methods*". عندما تحدد فئة، فإنك تحدد أسماء خصائصها وتعطي كود أساليبها.

تصحيح البرامج وصيانتها أسهل بكثير إذا كنت تستخدم التغليف أو الكبسلة "*encapsulation*". هذه هي الفكرة القائلة بأن الفئة توفر طرقًا معينة (الواجهة "*interface*") للكود الذي يستخدم كائناتها، وبالتالي لا يصل الكود الخارجي مباشرةً إلى هياكل البيانات الخاصة بهذه الكائنات. وبالتالي، فإن التصحيح أسهل

لأنك تعرف مكان البحث عن الأخطاء - الكود الوحيد الذي يغير هيا كل بيانات الكائن موجود داخل الفئة "class" - وتكون الصيانة أسهل لأنه يمكنك تبديل تطبيقات فئة دون تغيير الكود الذي يستخدم الفئة، مثل طالما أنك تحتفظ بنفس الواجهة.

ربما يتضمن أي تصميم غير بديهي موجه للكائنات الوراثة "*inheritance*". هذه طريقة لتعريف فئة جديدة بالقول إنها مثل فئة موجودة، ولكن بخصائص وطرق معينة جديدة أو متغيرة. تسمى الفئة الأصلية الفئة الفائقة "*superclass*" (أو الأصل -الأب- "parent" أو الفئة الأساسية "base class")، وتسمى الفئة الجديدة الفئة الفرعية "*subclass*" (أو الفئة المشتقة "derived class"). الوراثة هي شكل من أشكال إعادة استخدام التعليمات البرمجية - يتم إعادة استخدام كود الفئة الفائقة بدلاً من نسخها ولصقها في فئة فرعية. يتم تمرير أي تحسينات أو تعديلات على الفئة الفائقة تلقائيًا إلى الفئة الفرعية.

# إنشاء كائن

من الأسهل بكثير إنشاء (أو إنشاء مثيل "*instantiate*") كائنات واستخدامها بدلاً من تعريف فئات الكائن، لذلك قبل أن نناقش كيفية تعريف الفئات، دعنا نلقي نظرة على إنشاء الكائنات. لإنشاء كائن من فئة معينة، استخدم الكلمة الأساسية new:

```
$object = new Class;
```

بافتراض أنه تم تعريف فئة الشخص "Person"، فإليك كيفية إنشاء كائن الشخص:

```
$moana = new Person;
```

لا تقتبس اسم الفئة، وإلا ستحصل على خطأ في التجميع "compilation error":

```
$moana = new "Person"; // does not work
```

تسمح لك بعض الفئات بتمرير المدخلات إلى الاستدعاء. يجب أن توضح وثائق الفئة ما إذا كانت تقبل المدخلات. إذا حدث ذلك، فستنشئ كائنات مثل هذه:

```
$object = new Person("Sina", 35);
```

لا يلزم أن يكون اسم الفئة مضمّناً في برنامجك. يمكنك توفير اسم الفئة من خلال متغير:

```
$class = "Person";
$object = new $class;
// is equivalent to
```

```
$object = new Person;
```

يؤدي تحديد فئة غير موجودة إلى حدوث خطأ في وقت التشغيل.

المتغيرات التي تحتوي على مراجع الكائنات هي مجرد متغيرات عادية - يمكن استخدامها بنفس الطرق مثل المتغيرات الأخرى. لاحظ أن المتغيرات المتغيرة تعمل مع الكائنات، كما هو موضح هنا:

```
$account = new Account;
$object = "account";
${$object}->init(50000, 1.10); // same as $account->init
```

# الوصول إلى الخصائص والأساليب

بمجرد أن يكون لديك كائن، يمكنك استخدام الترميز -> للوصول إلى أساليب وخصائص الكائن:

```
$object->propertyname $object->methodname([arg, ... ])
```

كمثال:

```
echo "Moana is {$moana->age} years old.\n"; // property access
$moana->birthday(); // method call
$moana->setAge(21); // method call with arguments
```

تعمل الطرق بنفس الطريقة التي تعمل بها الدوال (فقط على وجه التحديد للكائن المعني)، حتى تتمكن من أخذ مدخلات وإرجاع قيمة:

```
$clan = $moana->family("extended");
```

ضمن تعريف الفصل، يمكنك تحديد الأساليب والخصائص التي يمكن الوصول إليها بشكل عام والتي لا يمكن الوصول إليها إلا من داخل الفئة نفسها باستخدام معدِّلات الوصول العامة والخاصة. يمكنك استخدام هذه لتوفير الكبسلة.

يمكنك استخدام المتغيرات المتغيرة مع أسماء الخصائص:

```
$prop = 'age';
echo $moana->$prop;
```

الأسلوب أو الطريقة الثابتة "static method" هي الطريقة التي يتم استدعاؤها على فئة وليس على كائن. مثل هذه الأساليب لا يمكن الوصول إلى الخصائص. اسم الطريقة الثابتة هو اسم الفئة متبوعًا بنقطتين واسم الدالة. على سبيل المثال، يستدعي هذا الأسلوب p() الثابت في فئة HTML:

```
HTML::p("Hello, world");
```

عند التصريح عن فئة، فإنك تحدد الخصائص والأساليب التي تكون ثابتة باستخدام خاصية الوصول الثابت.

بمجرد الإنشاء، يتم تمرير الكائنات عن طريق المرجع - أي، بدلاً من النسخ حول الكائن بأكمله (وهو مسعى يستهلك الوقت والذاكرة)، يتم تمرير مرجع إلى الكائن بدلاً من ذلك. فمثلا:

```
$f = new Person("Pua", 75);

$b = $f; // $b and $f point at same object
$b->setName("Hei Hei");

printf("%s and %s are best friends.\n", $b->getName(),
$f->getName());
Hei Hei and Hei Hei are best friends.
```

إذا كنت تريد إنشاء نسخة طبق الأصل من كائن ما، يمكنك استخدام عامل النسخ:

```
$f = new Person("Pua", 35);

$b = clone $f; // make a copy
$b->setName("Hei Hei");// change the copy
```

```
printf("%s and %s are best friends.\n", $b->getName(),
$f->getName());
```

```
Pua and Hei Hei are best friends.
```

عندما تستخدم عامل النسخ لإنشاء نسخة من كائن وتعلن هذه الفئة عن طريقة ()clone__، يتم استدعاء هذه الطريقة على الكائن الجديد فور استنساخه. يمكنك استخدام هذا في الحالات التي يحتفظ فيها الكائن بموارد خارجية (مثل مقابض الملفات "file handles") لإنشاء موارد جديدة، بدلاً من نسخ الموارد الموجودة.

# إعلان فئة

لتصميم برنامجك أو مكتبة الأكواد الخاصة بك بطريقة موجهة للكائنات، ستحتاج إلى تحديد الفئات الخاصة بك، باستخدام الكلمة الأساسية class. يتضمن تعريف الفئة اسم الفئة وخصائصها وطرقها. أسماء الفئات غير حساسة لحالة الأحرف ويجب أن تتوافق مع قواعد معرفات PHP. من بين أمور أخرى، اسم الفئة stdClass محجوز. في ما يلي بناء الجملة لتعريف فئة:

```
class classname [ extends baseclass ] [ implements interfacename ,
 [interfacename, ... ] ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) [: type ] {
 // code
 }
 ...
 ]
}
```

## طرق التصريح Declaring Methods

الطريقة هي دالة محددة داخل فئة. بالرغم من أن PHP لا تفرض قيودًا خاصة، إلا أن معظم التوابع تعمل فقط على البيانات الموجودة داخل الكائن الذي توجد فيه الطريقة. يمكن استخدام أسماء الطرق

التي تبدأ بشرطتين سفليتين (__) في المستقبل بواسطة PHP (ويتم استخدامها حالياً لطرق تسلسل الكائنات ()sleep__ و ()wakeup__، الموصوفة لاحقًا في هذا الفصل، من بين أمور أخرى)، لذلك يوصى بأن لا تبدأ أسماء الطرق الخاصة بك بهذا التسلسل.

ضمن الأسلوب، يحتوي المتغير this$ على مرجع للكائن الذي تم استدعاء الطريقة عليه. على سبيل المثال، إذا استدعيت ()moana->birthday$، داخل طريقة ()birthday، فإن this$ يحمل نفس قيمة moana$. تستخدم الطرق المتغير this$ للوصول إلى خصائص الكائن الحالي واستدعاء طرق أخرى على هذا الكائن.

إليك تعريف فئة بسيط لفئة الشخص يوضح المتغير this$ في العمل:

```
class Person {
 public $name = '';

 function getName() {
 return $this->name;
 }

 function setName($newName) {
 $this->name = $newName;
 }
}
```

كما ترى، أساليب getName() و setName() تستخدم $this للوصول إلى خاصية $name وتعيينها للكائن الحالي.

للإعلان عن طريقة كطريقة ثابتة، استخدم الكلمة الأساسية static. داخل الطرق الثابتة، لم يتم تعريف المتغير $this. فمثلا:

```
class HTMLStuff {
 static function startTable() {
 echo "<table border=\"1\">\n";
 }

 static function endTable() {
 echo "</table>\n";
 }
}

HTMLStuff::startTable();
 // print HTML table rows and columns
HTMLStuff::endTable();
```

إذا قمت بتعريف طريقة باستخدام الكلمة الأساسية final، فلن تتمكن الفئات الفرعية من تجاوز هذه الطريقة. فمثلا:

```
class Person {
 public $name;
```

```
 final function getName() {
 return $this->name;
 }
}

class Child extends Person {
 // syntax error
 function getName() {
 // do something
 }
}
```

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية الأساليب. يجب الإعلان عن الطرق التي يمكن الوصول إليها خارج الطرق على الكائن للعام "public"، يجب الإعلان عن التوابع الموجودة على مثيل والتي لا يمكن استدعاؤها إلا من خلال طرق ضمن نفس الفئة "private". أخيرًا، لا يمكن استدعاء العمليات التي تم الإعلان عنها على أنها محمية "protected" إلا من داخل عمليات فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. تحديد رؤية أساليب الفصل أمر اختياري؛ إذا لم يتم تحديد الرؤية، تكون الطريقة عامة. على سبيل المثال، قد تحدد:

```
class Person {
 public $age;

 public function __construct() {
 $this->age = 0;
 }
```

```
 public function incrementAge() {
 $this->age += 1;
 $this->ageChanged();
 }

 protected function decrementAge() {
 $this->age -= 1;
 $this->ageChanged();
 }

 private function ageChanged() {
 echo "Age changed to {$this->age}";
 }
}

class SupernaturalPerson extends Person {
 public function incrementAge() {
 // ages in reverse
 $this->decrementAge();
 }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge(); // not allowed
$person->ageChanged(); // also not allowed
```

```
$person = new SupernaturalPerson;
$person->incrementAge(); // calls decrementAge under the hood
```

يمكنك استخدام تلميح الكتابة (الموصوف في الفصل 3) عند تعريف طريق للكائن:

```
class Person {
 function takeJob(Job $job) {
 echo "Now employed as a {$job->title}\n";
 }
}
```

عندما تقوم إحدى الطرق بإرجاع قيمة، يمكنك استخدام تلميح الكتابة للإعلان عن نوع القيمة المرجعة للطريقة:

```
class Person {
 function bestJob(): Job {
 $job = Job("PHP developer");

 return $job;
 }
}
```

## تعريف الخصائص

في التعريف السابق لفئة الشخص، أعلنا صراحة عن خاصية $name. إعلانات الملكية اختيارية وهي ببساطة مجاملة لمن يحافظ على برنامجك. إنه أسلوب PHP جيد للإعلان عن خصائصك، ولكن يمكنك إضافة خصائص جديدة في أي وقت.

في ما يلي نسخة من فئة الشخص التي تحتوي على خاصية $name غير معرّفة:

```
class Person {
 function getName() {
 return $this->name;
 }

 function setName($newName) {
 $this->name = $newName;
 }
}
```

يمكنك تعيين قيم افتراضية للخصائص، لكن هذه القيم الافتراضية يجب أن تكون ثوابت بسيطة:

```
public $name = "J Doe"; // works
public $age = 0; // works
public $day = 60 * 60 * hoursInDay(); // doesn't work
```

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية الخصائص. يجب الإعلان عن الخصائص التي يمكن الوصول إليها خارج نطاق الكائن للعام "public"، يجب الإعلان عن الخصائص الموجودة على مثيل والتي يمكن الوصول إليها فقط من خلال طرق ضمن نفس "private" الفئة. أخيرًا، لا يمكن الوصول إلى

الخصائص التي تم الإعلان عنها على أنها محمية "protected" إلا من خلال طرق فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. على سبيل المثال، قد تعلن عن فئة مستخدم:

```
class Person {
 protected $rowId = 0;

 public $username = 'Anyone can see me';

 private $hidden = true;
}
```

بالإضافة إلى خصائص مثيلات الكائنات، تسمح لك PHP بتعريف الخصائص الثابتة، وهي متغيرات في فئة الكائن، ويمكن الوصول إليها عن طريق الرجوع إلى الخاصية باسم الفئة. فمثلا:

```
class Person {
 static $global = 23;
}

$localCopy = Person::$global;
```

داخل مثيل من فئة الكائن، يمكنك أيضًا الرجوع إلى الخاصية الثابتة باستخدام الكلمة الأساسية `self`، مثل: `echo self::$global;`.

إذا تم الوصول إلى خاصية على كائن غير موجود، وإذا تم تحديد طريقة ()__get أو ()__set لفئة الكائن، يتم منح هذه الطريقة فرصة إما لاسترداد قيمة أو تعيين قيمة لتلك الخاصية.

على سبيل المثال، قد تعلن عن فئة تمثل البيانات التي تم سحبها من قاعدة بيانات، ولكنك قد لا ترغب في سحب قيم بيانات كبيرة - مثل الكائنات الثنائية الكبيرة "Binary Large Objects" (BLOBs)- ما لم يتم طلب ذلك تحديدًا. تمثل إحدى طرق تنفيذ ذلك، بالطبع، في إنشاء طرق وصول للخاصية التي تقرأ وتكتب البيانات عند الطلب. هناك طريقة أخرى وهي استخدام طرق التحميل الزائد "overloading" هذه:

```
class Person {
 public function __get($property) {
 if ($property === 'biography') {
 $biography = "long text here..."; // would retrieve from database

 return $biography;
 }
 }

 public function __set($property, $value) {
 if ($property === 'biography') {
 // set the value in the database
 }
 }
}
```

## تعريف الثوابت

كما هو الحال مع الثوابت العامة، المعينة من خلال دالة ()define، توفر PHP طريقة لتعيين ثوابت داخل فئة. مثل الخصائص الثابتة، يمكن الوصول إلى الثوابت مباشرة من خلال الفئة أو ضمن طرق الكائن باستخدام التدوين الذاتي "self notation". بمجرد تحديد الثابت، لا يمكن تغيير قيمته:

```
class PaymentMethod {
 public const TYPE_CREDITCARD = 0;
 public const TYPE_CASH = 1;
}

echo PaymentMethod::TYPE_CREDITCARD;
0
```

كما هو الحال مع الثوابت العامة، من الشائع تعريف ثوابت الفئة بمعرفات الأحرف الكبيرة.

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية ثوابت الفئة. يجب إعلان ثوابت الفئة التي يمكن الوصول إليها خارج الطرق على الكائن لـ public، يجب إعلان ثوابت الفئة على مثيل لا يمكن الوصول إليه إلا من خلال طرق ضمن نفس الفئة private. أخيرًا، لا يمكن الوصول إلى الثوابت المعلنة على أنها protected إلا من داخل طرق فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. تحديد رؤية ثوابت الفئة اختياري؛ إذا لم يتم تحديد الرؤية، تكون الطريقة public. على سبيل المثال، قد تحدد:

```
class Person {
 protected const PROTECTED_CONST = false;
```

```
 public const DEFAULT_USERNAME = "<unknown>";
 private INTERNAL_KEY = "ABC1234";
}
```

**الوراثة "Inheritance"**

لترث الخصائص والطرق من فئة أخرى، استخدم الكلمات الأساسية extends في تعريف الفئة، متبوعة باسم الفئة الأساسية:

```
class Person {
 public $name, $address, $age;
}

class Employee extends Person {
 public $position, $salary;
}
```

تحتوي فئة الموظف "Employee" على خصائص الـ: $position والراتب $salary، بالإضافة إلى $name و $address وخصائص $age الموروثة من فئة الشخص "Person".

إذا كان للفئة المشتقة خاصية أو طريقة بنفس الاسم كواحد في فئتها الأصلية أو الفئة الأب "parent"، فإن الخاصية أو الطريقة في الفئة المشتقة لها الأسبقية على الخاصية أو الطريقة في الفئة الأصلية. إشارة "Referencing" الخاصية ترجع قيمة الخاصية على الطفل "child"، بينما الإشارة إلى الطريقة تستدعي الطريقة على الطفل.

استخدم parent::*method*() للوصول إلى طريقة تم تجاوزها في فئة الأب:

```
parent::birthday(); // call parent class's birthday() method
```

هناك خطأ شائع يتمثل في ترميز اسم الفئة الأصل في استدعاءات للطرق التي تم تجاوزها:

```
Creature::birthday(); // when Creature is the parent class
```

يعد هذا خطأً لأنه يوزع معرفة اسم الفصل الأصلي في جميع أنحاء الفصل المشتق. يؤدي استخدام ::parent إلى تمركز معرفة الفئة الأصل في جملة extends.

إذا كان من الممكن تصنيف طريقة ما ضمن فئة فرعية وتريد التأكد من أنك تستدعيها في الفئة الحالية، فاستخدم self::*method*():

```
self::birthday(); // call this class's birthday() method
```

للتحقق مما إذا كان الكائن هو مثيل لفئة معينة أو إذا كان يطبق واجهة معينة (راجع قسم "الواجهات")، يمكنك استخدام العامل instanceof:

```
if ($object instanceof Animal) {
 // do something
}
```

## واجهات

توفر الواجهات طريقة لتحديد العقود "contracts" التي تلتزم بها الطبقة، توفر الواجهة نماذج أولية وثوابت للطريقة، ويجب أن توفر أي فئة تنفذ الواجهة عمليات تنفيذ لجميع الأساليب في الواجهة. إليك بنية تعريف الواجهة:

```
interface interfacename {
 [ function functionname();
 ...
 ]
}
```

للإعلان عن تنفيذ فئة لواجهة، قم بتضمين الكلمة الأساسية implements وأي عدد من الواجهات، مفصولة بفواصل:

```
interface Printable {
 function printOutput();
}

class ImageComponent implements Printable {
 function printOutput() {
 echo "Printing an image...";
 }
}
```

قد ترث الواجهة من الواجهات الأخرى (بما في ذلك الواجهات المتعددة) طالما لم ترث أي من الواجهات من أساليب التصريح التي تحمل نفس الاسم مثل تلك المعلنة في الواجهة الفرعية.

## السمات Traits

توفر السمات آلية لإعادة استخدام الكود خارج التسلسل الهرمي للفئة. تسمح لك السمات بمشاركة الدوال عبر الفئات المختلفة التي لا تشترك (ولا ينبغي) في صنف مشترك في التسلسل الهرمي للفصل. في ما يلي بناء الجملة لتعريف سمة:

```
trait traitname [ extends baseclass ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) {
 // code
 }
 ...
 ]
}
```

للإعلان أن الفصل يجب أن يشتمل على طرق السمات، قم بتضمين الكلمة الأساسية use وأي عدد من السمات، مفصولة بفواصل:

```
trait Logger {
 public function log($logString) {
 $className = __CLASS__;
 echo date("Y-m-d h:i:s", time()) . ": [{$className}]
{$logString}";
 }
}
```

```
class User {
 use Logger;

 public $name;

 function __construct($name = '') {
 $this->name = $name;
 $this->log("Created user '{$this->name}'");
 }

 function __toString() {
 return $this->name;
 }
}

class UserGroup {
 use Logger;

 public $users = array();

 public function addUser(User $user) {
 if (!in_array($this->users, $user)) {
 $this->users[] = $user;
 $this->log("Added user '{$user}' to group");
 }
```

```
 }
}

$group = new UserGroup;
$group->addUser(new User("Franklin"));
```

**2012-03-09 07:12:58: [User] Created user 'Franklin'2012-03-09 07:12:58:**

**[UserGroup] Added user 'Franklin' to group**

الطرق التي تحددها سمة المسجل "Logger" متاحة لحالات فئة UserGroup كما لو كانت محددة في تلك الفئة.

للإعلان عن أن السمة يجب أن تتكون من سمات أخرى، قم بتضمين جملة use في إعلان السمة، متبوعاً بواحد أو أكثر أسماء السمات مفصولة بفواصل، كما هو موضح هنا:

```
trait First {
 public function doFirst( {
 echo "first\n";
  }
}

trait Second {
 public function doSecond() {
 echo "second\n";
  }
}
```

```
trait Third {
 use First, Second;

 public function doAll() {
 $this->doFirst();
 $this->doSecond();
 }
}

class Combined {
 use Third;
}

$object = new Combined;
$object->doAll();
firstsecond
```

يمكن للسمات أن تعلن عن طرق مجردة "abstract methods".

إذا كان الفصل "class" يستخدم سمات متعددة تحدد نفس الطريقة، فإن PHP تعطي خطأ فادحًا. ومع ذلك، يمكنك تجاوز هذا السلوك بإخبار المترجم بالتحديد أي تطبيق تريد استخدامه لطريقة معينة. عند تحديد السمات التي يتضمنها الفصل، استخدم الكلمة الأساسية insteadof لكل تعارض "conflict":

```
trait Command {
 function run() {
 echo "Executing a command\n";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon\n";
 }
}

class Person {
 use Command, Marathon {
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
Running a marathon
```

بدلاً من اختيار طريقة واحدة فقط لتضمينها، يمكنك استخدام الكلمة الرئيسية as لتسمية مستعارة "alias" لطريقة سمة داخل الفئة بما في ذلك اسم مختلف. لا يزال يتعين عليك حل أي تعارضات في السمات المضمنة صراحة. فمثلا:

```
trait Command {
 function run() {
 echo "Executing a command";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon";
 }
}

class Person {
 use Command, Marathon {
 Command::run as runCommand;
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
$person->runCommand();
```

**Running a marathonExecuting a command**

## طرق مجردة "Abstract Methods"

توفر PHP أيضًا آلية للإعلان عن وجوب تنفيذ عمليات معينة في الفئة بواسطة الفئات الفرعية "subclasses" - لم يتم تحديد تنفيذ هذه الأساليب في الفئة الرئيسية. في هذه الحالات، تقدم طريقة مجردة، بالإضافة إلى ذلك، إذا احتوت الفئة على أي عمليات مُعرّفة "defined" على أنها مجردة "abstract"، فيجب أيضًا إعلان الفئة كفئة مجردة:

```
abstract class Component {
 abstract function printOutput();
}

class ImageComponent extends Component {
 function printOutput() {
 echo "Pretty picture";
 }
}
```

لا يمكن إنشاء مثيل للفئات المجردة. لاحظ أيضًا أنه على عكس بعض اللغات، لا تسمح لك PHP بتوفير تنفيذ "implementation" افتراضي للطرق المجردة.

يمكن للسمات أيضًا أن تعلن عن طرق مجردة. يجب أن تطبق الفئات التي تتضمن سمة تحدد طريقة مجردة تلك الطريقة:

```
trait Sortable {
 abstract function uniqueId();
```

```
 function compareById($object) {
 return ($object->uniqueId() < $this->uniqueId()) ? -1
: 1;
 }
}

class Bird {
 use Sortable;

 function uniqueId() {
 return __CLASS__ . ":{$this->id}";
 }
}

// this will not compile
class Car {
 use Sortable;
}

$bird = new Bird;
$car = new Car;
$comparison = $bird->compareById($car);
```

عند تنفيذ طريقة مجردة في فئة فرعية، يجب أن تتطابق تواقيع "signatures" الطريقة - أي، يجب أن تأخذ نفس العدد من المعلمات المطلوبة، وإذا كان لأي من المعلمات تلميحات كتابة، فيجب أن تتطابق تلميحات الكتابة هذه. بالإضافة إلى ذلك، يجب أن يكون للطريقة نفس الرؤية المقيدة أو أقل.

**Constructors**

يمكنك أيضًا تقديم قائمة بالمدخلات التي تتبع اسم الفئة عند إنشاء كائن:

```
$person = new Person("Fred", 35);
```

يتم تمرير هذه المدخلات إلى منشئ "*constructor*" الفئة، وهي دالة خاصة تقوم بتهيئة خصائص الفئة.

المُنشئ "constructor" هو دالة في الفئة تسمى ()construct__. فيما يلي مُنشئ لفئة Person:

```
class Person {
 function __construct($name, $age) {
 $this->name = $name;
 $this->age = $age;
 }
}
```

لا توفر PHP سلسلة تلقائية للمُنشئين؛ بمعنى، إذا قمت بإنشاء مثيل لفئة مشتقة، فسيتم استدعاء المُنشئ فقط في الفئة المشتقة تلقائيًا. لكي يتم استدعاء مُنشئ الفئة الأصلية، يجب على المُنشئ في الفئة المشتقة استدعاء المُنشئ صراحةً. في هذا المثال، يستدعي مُنشئ فئة Employee مُنشئ Person:

```
class Person {
 public $name, $address, $age;

 function __construct($name, $address, $age) {
 $this->name = $name;
 $this->address = $address;
```

```
 $this->age = $age;
 }
}

class Employee extends Person {
 public $position, $salary;

 function __construct($name, $address, $age, $position,
$salary) {
 parent::__construct($name, $address, $age);

 $this->position = $position;
 $this->salary = $salary;
 }
}
```

**المدمرات "Destructors"**

عندما يتم تدمير كائن، مثل عند إزالة المرجع الأخير إلى كائن أو الوصول إلى نهاية البرنامج النصي، يتم استدعاء أداة التدمير "*destructor*" الخاصة به. نظراً لأن PHP تقوم تلقائيًا بتنظيف جميع الموارد عندما تقع خارج النطاق وفي نهاية تنفيذ البرنامج النصي، فإن تطبيقها يكون محدوداً. المدمر هو طريقة تسمى __destruct():

```
class Building {
 function __destruct() {
 echo "A Building is being destroyed!";
 }
}
```

# فئات مجهولة "Anonymous Classes"

أثناء إنشاء كائنات وهمية "mock" للاختبار، من المفيد إنشاء فئات مجهولة. تتصرف الفئة المجهولة بنفس سلوك أي فئة أخرى، باستثناء أنك لا تقدم اسمًا (مما يعني أنه لا يمكن إنشاء مثيل له بشكل مباشر):

```
class Person {
 public $name = '';

 function getName() {
 return $this->name;
 }
}

// return an anonymous implementation of Person
$anonymous = new class() extends Person {
 public function getName() {
 // return static value for testing purposes
 return "Moana";
 }
}; // note: requires closing semicolon, unlike
nonanonymous class definitions
```

على عكس مثيلات الفئات المسماة، لا يمكن إجراء تسلسل لمثيلات الفئات المجهولة. تؤدي محاولة إجراء تسلسل لمثيل لفئة مجهولة إلى حدوث خطأ.

# الاستبطان Introspection

الاستبطان هو قدرة البرنامج على فحص خصائص الكائن، مثل الاسم والفئة الأصلية (إن وجدت) والخصائص والطرق. باستخدام الاستبطان، يمكنك كتابة التعليمات البرمجية التي تعمل على أي فئة أو كائن. لست بحاجة لمعرفة الأساليب أو الخصائص المحددة عند كتابة التعليمات البرمجية الخاصة بك؛ بدلاً من ذلك، يمكنك اكتشاف تلك المعلومات في وقت التشغيل، مما يجعل من الممكن لك كتابة مصححات أخطاء "debuggers" عامة ومسلسلات "serializers" وملفات تعريف "profilers" وما شابه ذلك. في هذا القسم، نلقي نظرة على دوال الاستبطان التي توفرها PHP.

## اختبار الفئات "Examining Classes"

لتحديد ما إذا كانت الفئة موجودة أم لا، استخدم الدالة ()class_exists، والتي تأخذ سلسلة وترجع قيمة منطقية. بالتناوب، يمكنك استخدام دالة ()get_declared_classes، التي تُرجع مصفوفة من الفئات المحددة وتحقق مما إذا كان اسم الفئة موجوداً في المصفوفة التي تم إرجاعها:

```
$doesClassExist = class_exists(classname);
```

```
$classes = get_declared_classes();
$doesClassExist = in_array(classname, $classes);
```

يمكنك الحصول على الطرق والخصائص الموجودة في الفصل الدراسي (بما في ذلك تلك الموروثة من الفئات الفائقة) باستخدام الدالتين ()get_class_methods و ()get_class_vars. تأخذ هذه الدوال اسم فئة وتعيد مصفوفة:

```
$methods = get_class_methods(classname);
```

```
$properties = get_class_vars(classname);
```

يمكن أن يكون اسم الفئة إما متغيرًا يحتوي على اسم الفئة، أو bare word، أو سلسلة مقتبسة:

```
$class = "Person";
$methods = get_class_methods($class);
$methods = get_class_methods(Person); // same
$methods = get_class_methods("Person"); // same
```

المصفوفة التي تم إرجاعها بواسطة ()get_class_methods هي قائمة بسيطة بأسماء الطرق. المصفوفة الترابطية التي تم إرجاعها بواسطة ()get_class_vars تعين أسماء الخصائص إلى القيم وتتضمن أيضًا الخصائص الموروثة.

تمثل إحدى مشكلات ()get_class_vars في أنها تعرض فقط الخصائص التي لها قيم افتراضية وتكون مرئية في النطاق الحالي؛ لا توجد طريقة لاكتشاف الخصائص غير المهيأة.

استخدم ()get_parent_class للعثور على الفئة الأصلية للفئة:

```
$superclass = get_parent_class(classname);
```

يسرد المثال 6-1 دالة ()displayClasses، والتي تعرض جميع الفئات المُعلنة حاليًا وطرق وخصائص كل منها.

مثال 6-1. عرض جميع الفئات المصرح عنها

```
function displayClasses() {
 $classes = get_declared_classes();

 foreach ($classes as $class) {
 echo "Showing information about {$class}<br />";
 $reflection = new ReflectionClass($class);

 $isAnonymous = $reflection->isAnonymous() ? "yes" :
"no";
 echo "Is Anonymous: {$isAnonymous}<br />";

 echo "Class methods:<br />";
 $methods = $reflection-
>getMethods(ReflectionMethod::IS_STATIC);

 if (!count($methods)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach ($methods as $method) {
 echo "<b>{$method}</b>()<br />";
 }
 }
```

```
echo "Class properties:<br />";

$properties = $reflection->getProperties();

if (!count($properties)) {
echo "<i>None</i><br />";
}
else {
foreach(array_keys($properties) as $property) {
echo "<b>\${$property}</b><br />";
}
}

echo "<hr />";
}
}
```

## اختبار الكائن

للحصول على الفئة التي ينتمي إليها الكائن، تأكد أولاً من أنه كائن يستخدم الدالة is_object()، ثم احصل على الفئة باستخدام دالة get_class():

```
$isObject = is_object(var);
$classname = get_class(object);
```

قبل استدعاء طريقة على كائن، يمكنك التأكد من وجودها باستخدام الدالة method_exists():

```
$methodExists = method_exists(object, method);
```

استدعاء طريقة غير محددة يؤدي إلى استثناء وقت التشغيل.

تمامًا كما تُرجع الدالة get_class_vars() مجموعة من الخصائص لفئة ما، فإن get_object_vars() تُرجع مصفوفة من الخصائص المعينة في كائن:

```
$array = get_object_vars(object);
```

ومثلما يقوم get_class_vars() بإرجاع تلك الخصائص ذات القيم الافتراضية فقط، فإن get_object_vars() يعرض فقط تلك الخصائص التي تم تعيينها:

```
class Person {
 public $name;
 public $age;
}

$fred = new Person;
$fred->name = "Fred";
$props = get_object_vars($fred); // array('name' => "Fred", 'age' => NULL);
```

تقبل الدالة get_parent_class() إما كائنًا أو اسم فئة. تقوم بإرجاع اسم الفئة الأصل، أو FALSE إذا لم يكن هناك فئة أصل:

```
class A {}
class B extends A {}

$obj = new B;
```

```
echo get_parent_class($obj);
echo get_parent_class(B);
AA
```

## عينة برنامج الاستبطان "Sample Introspection Program"

يوضح المثال 6-2 مجموعة من الدوال التي تعرض صفحة مرجعية للمعلومات حول خصائص الكائن وطرقه وشجرة الوراثة.

مثال 6-2. دوال استبطان الكائن

```
// return an array of callable methods (include inherited methods)
function getCallableMethods($object): Array {
 $reflection = new ReflectionClass($object);
 $methods = $reflection->getMethods();

 return $methods;
}

// return an array of superclasses
function getLineage($object): Array {
 $reflection = new ReflectionClass($object);

 if ($reflection->getParentClass()) {
 $parent = $reflection->getParentClass();
```

```
 $lineage = getLineage($parent);
 $lineage[] = $reflection->getName();
 }
 else {
 $lineage = array($reflection->getName());
 }

 return $lineage;
}

// return an array of subclasses
function getChildClasses($object): Array {
 $reflection = new ReflectionClass($object);

 $classes = get_declared_classes();

 $children = array();

 foreach ($classes as $class) {
 $checkedReflection = new ReflectionClass($class);

 if      ($checkedReflection->isSubclassOf($reflection-
>getName())) {
 $children[] = $checkedReflection->getName();
 }
 }
```

```
 return $children;
}

// return an array of properties
function getProperties($object): Array {
 $reflection = new ReflectionClass($object);

 return $reflection->getProperties();
}

// display information on an object
function printObjectInfo($object) {
 $reflection = new ReflectionClass($object);
 echo "<h2>Class</h2>";
 echo "<p>{$reflection->getName()}</p>";

 echo "<h2>Inheritance</h2>";

 echo "<h3>Parents</h3>";
 $lineage = getLineage($object);
 array_pop($lineage);

 if (count($lineage) > 0) {
 echo "<p>" . join(" -&gt; ", $lineage) . "</p>";
 }
 else {
```

```
echo "<i>None</i>";
}

echo "<h3>Children</h3>";
$children = getChildClasses($object);
echo "<p>";

if (count($children) > 0) {
echo join(', ', $children);
}
else {
echo "<i>None</i>";
}

echo "</p>";

echo "<h2>Methods</h2>";
$methods = getCallableMethods($object);

if (!count($methods)) {
echo "<i>None</i><br />";
}
else {
foreach($methods as $method) {
echo "<b>{$method}</b>();<br />";
}
```

```
 }

 echo "<h2>Properties</h2>";
 $properties = getProperties($object);

 if (!count($properties)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach(array_keys($properties) as $property) {
 echo "<b>\${$property}</b> = " . $object->$property .
"<br />";
 }
 }

 echo "<hr />";
}
```

فيما يلي بعض نماذج الفئات والكائنات التي تمارس دوال الاستبطان من المثال 6-2:

```
class A {
 public $foo = "foo";
 public $bar = "bar";
 public $baz = 17.0;

 function firstFunction() { }
```

```
 function secondFunction() { }
}

class B extends A {
 public $quux = false;

 function thirdFunction() { }
}

class C extends B { }

$a = new A();
$a->foo = "sylvie";
$a->bar = 23;

$b = new B();
$b->foo = "bruno";
$b->quux = true;

$c = new C();

printObjectInfo($a);
printObjectInfo($b);
printObjectInfo($c);
```

# التسلسل "Serialization"

تسلسل "*Serializing*" الكائن يعني: تحويله إلى تمثيل bytestream يمكن تخزينه في ملف. هذا مفيد للبيانات المستمرة؛ على سبيل المثال، تقوم جلسات PHP بحفظ الكائنات واستعادتها تلقائيًا. التسلسل في PHP تلقائي في الغالب - يتطلب منك القليل من العمل الإضافي، بخلاف استدعاء الدالتين serialize() و unserialize():

```
$encoded = serialize(something);
$something = unserialize(encoded);
```

يتم استخدام التسلسل بشكل شائع مع جلسات PHP، والتي تتعامل مع التسلسل نيابة عنك. كل ما عليك فعله هو إخبار PHP بالمتغيرات التي يجب تتبعها، ويتم الاحتفاظ بها تلقائيًا بين الزيارات إلى صفحات موقعك. ومع ذلك، فإن الجلسات ليست الاستخدام الوحيد للتسلسل - إذا كنت تريد تنفيذ النموذج الخاص بك من الكائنات الثابتة "persistent objects"، فإن serialize() وunserialize() هما خياران طبيعيان.

يجب تعريف فئة الكائن قبل أن يحدث إلغاء التسلسل. تؤدي محاولة إلغاء تسلسل كائن لم يتم تعريف صنفه بعد إلى وضع الكائن في فئة stdClass، مما يجعله عديم الفائدة تقريبًا. إحدى النتائج العملية لذلك هي أنه إذا كنت تستخدم جلسات PHP لتسلسل الكائنات وإلغاء تسلسلها تلقائيًا، فيجب عليك تضمين الملف الذي يحتوي على تعريف فئة الكائن في كل صفحة على موقعك. على سبيل المثال، قد تبدأ صفحاتك على النحو التالي:

```
include "object_definitions.php"; // load object definitions
session_start(); // load persistent variables
?>
<html>...
```

PHP لها خطافان "hooks" للكائنات أثناء عملية التسلسل وإلغاء التسلسل: ()__sleep و ()__wakeup. تُستخدم هذه الطرق لإعلام الكائنات بأنها متسلسلة أو غير متسلسلة. يمكن إجراء تسلسل للكائنات إذا لم يكن لديها هذه الطرق؛ ومع ذلك، لن يتم إشعارهم بشأن العملية.

يتم استدعاء طريقة ()__sleep على كائن قبل التسلسل مباشرة؛ يمكنه إجراء أي تنظيف ضروري للحفاظ على حالة الكائن، مثل إغلاق اتصالات قاعدة البيانات، وكتابة البيانات الدائمة غير المحفوظة، وما إلى ذلك. يجب أن تعيد مصفوفة تحتوي على أسماء أعضاء البيانات التي يجب كتابتها في سلسلة bytestream. إذا قمت بإرجاع مصفوفة فارغة، فلن تتم كتابة أي بيانات.

على العكس من ذلك، يتم استدعاء طريقة ()__wakeup على كائن مباشرة بعد إنشاء كائن من سلسلة bytestream. يمكن للطريقة اتخاذ أي إجراء تتطلبه، مثل إعادة فتح اتصالات قاعدة البيانات ومهام التهيئة الأخرى.

المثال 6-3 عبارة عن فئة كائن، Log، توفر طريقتين مفيدتين: ()write لإلحاق رسالة بملف السجل، و()read لجلب المحتويات الحالية لملف السجل. يستخدم ()__wakeup لإعادة فتح ملف السجل و ()__sleep لإغلاق ملف السجل.

مثال 6-3. ملف Log.php

```
class Log {
 private $filename;
 private $fh;

 function __construct($filename) {
 $this->filename = $filename;
 $this->open();
 }

 function open() {
 $this->fh = fopen($this->filename, 'a') or die("Can't
open {$this->filename}");
 }

 function write($note) {
 fwrite($this->fh, "{$note}\n");
 }

 function read() {
 return join('', file($this->filename));
 }

 function __wakeup(array $data): void {
 $this->filename = $data["filename"];
```

```
 $this->open();
 }

 function __sleep() {
 // write information to the account file
 fclose($this->fh);

 return ["filename" => $this->filename];
 }
}
```

قم بتخزين تعريف فئة Log في ملف يسمى *Log.php*. تستخدم صفحة HTML الأمامية في المثال 6-4 فئة log وجلسات PHP لإنشاء متغير log دائم، وهو $logger.

مثال 6-4. *front.php*

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Front Page</title></head>
<body>

<?php
$now = strftime("%c");
```

```
if (!isset($_SESSION['logger'])) {
 $logger = new Log("/tmp/persistent_log");
 $_SESSION['logger'] = $logger;
 $logger->write("Created $now");

 echo("<p>Created session and persistent log object.</p>");
}
else {
 $logger = $_SESSION['logger'];
}

$logger->write("Viewed first page {$now}");

echo "<p>The log contains:</p>";
echo nl2br($logger->read());
?>

<a href="next.php">Move to the next page</a>

</body></html>
```

يوضح المثال 6-5 الملف next.php، صفحة HTML. يؤدي اتباع الرابط من الصفحة الأولى إلى هذه الصفحة إلى بدء تحميل الكائن الثابت $logger. يقوم استدعاء ()__wakeup بإعادة فتح ملف السجل بحيث يكون الكائن جاهزاً للاستخدام.

مثال 6-5. next.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Next Page</title></head>
<body>

<?php
$now = strftime("%c");
$logger = $_SESSION['logger'];
$logger->write("Viewed page 2 at {$now}");

echo "<p>The log contains:";
echo nl2br($logger->read());
echo "</p>";
?>

</body></html>
```

# مالتالي

تعلم كيفية استخدام الكائنات في البرامج النصية الخاصة بك هو مهمة هائلة. في الفصل التالي، ننتقل من دلالات اللغة إلى الممارسة ونعرض لك واحدة من أكثر فئات PHP شيوعًا استخدامًا من الفئات الموجهة للكائنات - فئات التاريخ والوقت.

# الفصل السابع: التاريخ والوقت

من المحتمل أن يكون مطور PHP النموذجي على دراية بدوال التاريخ والوقت المتاحة، مثل عند إضافة طابع تاريخ إلى إدخال سجل قاعدة البيانات أو حساب الفرق بين تاريخين. توفر PHP فئة DateTime التي يمكنها معالجة معلومات التاريخ والوقت في وقت واحد، بالإضافة إلى فئة DateTimeZone التي تعمل جنبًا إلى جنب معها.

أصبحت إدارة المنطقة الزمنية أكثر بروزًا في السنوات الأخيرة مع ظهور بوابات الويب ومجتمعات الويب الاجتماعية مثل Facebook و Twitter. لتكون قادرًا على نشر المعلومات على موقع ويب وجعلها تتعرف على مكانك في العالم فيما يتعلق بالآخرين على نفس الموقع هو بالتأكيد مطلب هذه الأيام. ومع ذلك، ضع في اعتبارك أن دالة مثل ()date تأخذ المعلومات الافتراضية من الخادم الذي يعمل عليه البرنامج النصي، لذلك ما لم يخبرك العملاء البشريون عن مكانهم في العالم، فقد يكون من الصعب تحديد موقع المنطقة الزمنية تلقائيا. بمجرد أن تعرف المعلومات، يصبح من السهل معالجة تلك البيانات (المزيد عن المناطق الزمنية لاحقًا في هذا الفصل).

**ملاحظة:**

تحتوي دوال التاريخ الأصلي (والمرتبطة به) على خطأ توقيت في Windows وبعض عمليات تثبيت Unix. لا يمكنهم معالجة التواريخ قبل 13 ديسمبر 1901 أو ما بعد 19 يناير 2038، نظرًا لطبيعة العدد الصحيح المكون من 32bit والمستخدم لإدارة بيانات التاريخ والوقت. لذلك، يوصى باستخدام فئة DateTime الأحدث للحصول على دقة أفضل في المستقبل.

هناك أربع فئات مترابطة للتعامل مع التواريخ والأوقات. فئة DateTime تتعامل مع التواريخ نفسها، تعالج فئة DateTimeZone المناطق الزمنية، تعالج فئة DateInterval فترات زمنية بين مثيلين من DateTime، وأخيرًا، تتعامل فئة DatePeriod مع الاجتياز عبر فترات زمنية منتظمة من التواريخ والأوقات. هناك نوعان من الفئات الداعمة الأخرى التي نادرًا ما يتم استخدامها تسمى DateTimeImmutable و DateTimeInterface والتي تعد جزءًا من "عائلة" DateTime بأكملها، لكننا لن نغطي تلك الموجودة في هذا الفصل.

من الطبيعي أن يكون مُنشئ فئة DateTime هو المكان الذي يبدأ منه كل شيء. تأخذ هذه الطريقة معلمتين، الطابع الزمني والمنطقة الزمنية. فمثلا:

```
$dt = new DateTime("2019-06-27 16:42:33", new DateTimeZone("America/Halifax"));
```

أنشأنا الكائن dt$، وخصصنا له سلسلة التاريخ والوقت بالمعامل الأول، وضبطنا المنطقة الزمنية بالمعامل الثاني. هنا، نعمل على إنشاء مثيل DateTimeZone مضمنًا، ولكن يمكنك بالتناوب إنشاء مثيل لكائن DateTimeZone في متغيره الخاص ثم استخدامه في المُنشئ، مثل:

```
$dtz = new DateTimeZone("America/Halifax");
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

من الواضح الآن أننا نقوم بتعيين قيم مضمنة لهذه الفئات، وقد لا يكون هذا النوع من المعلومات متاحًا دائمًا للكود الخاص بك أو قد لا يكون ما تريده. بدلاً من ذلك، يمكننا التقاط قيمة المنطقة الزمنية من الخادم واستخدامها داخل فئة DateTimeZone. لالتقاط قيمة الخادم الحالية، استخدم كودا مشابهًا لما يلي:

```
$tz = ini_get('date.timezone');
```

```
$dtz = new DateTimeZone($tz);
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

تؤسس أمثلة هذا الكود مجموعة من القيم لفئتين، DateTime وDateTimeZone. في النهاية، سوف تستخدم هذه المعلومات بطريقة ما في مكان آخر في البرنامج النصي الخاص بك. إحدى طرق فئة DateTime تسمى ()format، وهي تستخدم نفس أكواد إخراج التنسيق مثل دالة ()date_format. في قسم الدالة ()date_format. فيما يلي عينة من طريقة ()format التي يتم إرسالها إلى المتصفح كإخراج:

```
echo "date: " . $dt->format("Y-m-d h:i:s");
date: 2019-06-27 04:42:33
```

لقد قدمنا حتى الآن التاريخ والوقت للمنشئ، ولكن في بعض الأحيان قد ترغب أيضًا في التقاط قيم التاريخ والوقت من الخادم. للقيام بذلك، ما عليك سوى توفير السلسلة "now" كمعامل أول.

الكود التالي يفعل نفس الأمثلة الأخرى، باستثناء أننا نحصل هنا على قيم فئة التاريخ والوقت من الخادم. في الواقع، نظرًا لأننا نحصل على المعلومات من الخادم، فإن خصائص الفئة مليئة بشكل كامل (لاحظ أن بعض مثيلات PHP لن تحتوي على هذه المعلمة، وبالتالي ستعرض خطأ، وقد لا تتطابق المنطقة الزمنية للخادم مع خاصة):

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);
$dt = new DateTime("now", $dtz);

echo "date: " . $dt->format("Y-m-d h:i:s");
```

**date: 2019-06-27 04:02:54**

تقوم طريقة ()diff الخاصة بـ DateTime بما قد نتوقعه — فهي تعرض الفرق بين تاريخين. القيمة المرجعة للطريقة هي مثيل لفئة DateInterval.

لمعرفة الفرق بين مثيلي DateTime، استخدم:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

$past = new DateTime("2019-02-12 16:42:33", $dtz);
$current = new DateTime("now", $dtz);

// creates a new instance of DateInterval
$diff = $past->diff($current);

$pastString = $past->format("Y-m-d");
$currentString = $current->format("Y-m-d");
$diffString = $diff->format("%yy %mm, %dd");

echo "Difference between {$pastString} and {$currentString} is {$diffString}";
```

**Difference between 2019-02-12 and 2019-06-27 is 0y 4m, 14d**

يتم استدعاء طريقة ()diff على أحد كائنات DateTime مع تمرير كائن DateTime آخر كمعامل. ثم نقوم بإعداد إخراج المتصفح باستخدام استدعاءات طريقة ()format.

لاحظ أن فئة DateInterval لها طريقة ()format أيضًا. نظرًا لأنه يتعامل مع الاختلاف بين تاريخين، فإن أكواد أحرف التنسيق تختلف قليلاً عن تلك الخاصة بفئة DateTime. يسبق كل كود حرف بعلامة النسبة المئوية ٪. يتم توفير أكواد الأحرف المتاحة في الجدول 7-1.

الجدول 7-1. DateInterval تنسيق أحرف التحكم

| الحرف | تأثير التنسيق |
|---|---|
| a | عدد الأيام (على سبيل المثال: 23) |
| d | عدد الأيام غير المدرجة بالفعل في عدد الأشهر |
| D | عدد الأيام، بما في ذلك الصفر البادئ إذا كان أقل من 10 أيام (على سبيل المثال: 02 و 125) |
| f | عدد ميكرو ثانية (على سبيل المثال: 6602 أو 41569) |
| F | عدد ميكرو ثانية رقمية بصفر بادئ، ستة أرقام على الأقل في الطول (على سبيل المثال: 006602 أو 041569) |
| h | عدد الساعات |

| الحرف | تأثير التنسيق |
|---|---|
| H | عدد الساعات، بما في ذلك الصفر البادئ إذا كان أقل من 10 ساعات (على سبيل المثال: 12 و 04) |
| i | عدد الدقائق |
| I | عدد الدقائق، بما في ذلك الصفر البادئ إذا كان أقل من 10 دقائق (على سبيل المثال: 05 و 33) |
| m | عدد الأشهر |
| M | عدد الأشهر، بما في ذلك الصفر البادئ إذا كان أقل من 10 أشهر (على سبيل المثال، 05 و 1533) |
| R | - إذا كان الفرق سالبا، فارغة إذا كان الفرق موجبا |
| R | - إذا كان الفرق سالبا، + إذا كان الفرق موجبا |
| s | عدد الثواني |
| S | عدد الثواني، بما في ذلك الصفر البادئ إذا كانت أقل من 10 ثواني (على سبيل المثال: 15 و 05) |
| y | عدد السنوات |

| الحرف | تأثير التنسيق |
| --- | --- |
| Y | عدد السنوات، بما في ذلك الصفر البادئ إذا كان أقل من 10 سنوات (على سبيل المثال: 00 و 12) |
| % | حرف % |

دعونا نلقي نظرة عن كثب على فئة DateTimeZone الآن. يمكن رفع إعداد المنطقة الزمنية من ملف *php.ini* باستخدام get_ini(). يمكنك الحصول على مزيد من المعلومات من كائن المنطقة الزمنية باستخدام طريقة getLocation(). يوفر البلد الأصلي للمنطقة الزمنية وخط الطول وخط العرض بالإضافة إلى بعض التعليقات. باستخدام هذه الأسطر القليلة من التعليمات البرمجية، يمكنك الحصول على بدايات نظام GPS قائم على الويب:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

echo "Server's Time Zone: {$tz}<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
Server's Time Zone: America/Halifax
country_code CA
latitude 44.65
longitude -63.6
```

```
comments Atlantic - NS (most areas); PE
```

إذا كنت ترغب في تعيين منطقة زمنية غير منطقة الخادم، فيجب عليك تمرير هذه القيمة إلى مُنشئ كائن DateTimeZone. يعيّن هذا المثال المنطقة الزمنية لروما، إيطاليا، ويعرض المعلومات باستخدام طريقة getLocation():

```
$dtz = new DateTimeZone("Europe/Rome");

echo "Time Zone: " . $dtz->getName() . "<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
```

```
Time Zone: Europe/Rome
country_code IT
latitude 41.9
longitude 12.48333
comments
```

يمكن العثور على قائمة بأسماء المناطق الزمنية الصالحة حسب المناطق العالمية في دليل PHP على الإنترنت.

باستخدام هذه التقنية نفسها، يمكنك جعل موقع ويب "محلياً" للزائر من خلال توفير قائمة بالمناطق الزمنية المدعومة للزائر للاختيار من بينها ثم ضبط إعدادات php.ini مؤقتاً باستخدام دالة ini_set() طوال مدة الزيارة.

في حين أن هناك قدرًا معقولاً من قوة معالجة التاريخ والوقت التي توفرها الفئات التي ناقشناها في هذا الفصل، إلا أنها مجرد قمة جبل الجليد الذي يضرب به المثل. تأكد من قراءة المزيد حول هذه الفصول الدراسية وما يمكنهم فعله على موقع PHP الإلكتروني.

# مالتالي

هناك أكثر من مجرد إدارة للتواريخ يجب فهمها عند تصميم مواقع الويب داخل PHP، ونتيجة لذلك، هناك العديد من المشكلات التي يمكن أن تسبب لك التوتر وتزيد من عامل PITA (ألم في المؤخرة "**p**ain **i**n **t**he **a**ss"). يقدم الفصل التالي نصائح وحيلًا متعددة، بالإضافة إلى بعض "المشاكل" التي يجب الانتباه إليها، للمساعدة في تقليل نقاط الألم هذه. من بين الموضوعات التي يتم تناولها، تقنيات العمل مع المتغيرات، وإدارة بيانات النموذج، واستخدام أمان بيانات الويب SSL (طبقة مآخذ التوصيل الآمنة). اربط حزام الأمان! (Buckle up!)

# الفصل الثامن: تقنيات الويب

تم تصميم PHP كلغة برمجة نصية على الويب، وعلى الرغم من أنه من الممكن استخدامها في نصوص سطر الأوامر وواجهة المستخدم الرسومية البحتة، إلا أن الويب يمثل الغالبية العظمى من استخدامات PHP. قد يحتوي موقع الويب الحيوي على نماذج وجلسات وأحيانًا إعادة توجيه، وهذا الفصل يشرح كيفية تنفيذ هذه العناصر في PHP. ستتعرف على كيفية توفير PHP للوصول إلى معلمات النماذج والملفات التي تم تحميلها، وكيفية إرسال ملفات تعريف الارتباط وإعادة توجيه المتصفح، وكيفية استخدام جلسات PHP، والمزيد.

## أساسيات HTTP

يعمل الويب على بروتوكول HTTP أو بروتوكول نقل النص التشعبي "Hypertext Transfer Protocol". يتحكم هذا البروتوكول في كيفية طلب متصفحات الويب للملفات من خوادم الويب وكيفية قيام الخوادم بإعادة إرسال الملفات. لفهم التقنيات المختلفة التي سنعرضها لك في هذا الفصل، يجب أن يكون لديك فهم أساسي لـ HTTP. للحصول على مناقشة أكثر شمولاً عن HTTP، راجع HTTP Pocket Reference (O'Reilly) بواسطة Clinton Wong.

عندما يطلب مستعرض الويب صفحة ويب، فإنه يرسل رسالة طلب HTTP إلى خادم ويب. تتضمن رسالة الطلب دائمًا بعض معلومات الرأس "header information"، وفي بعض الأحيان تتضمن أيضًا

نصًا "body". يستجيب خادم الويب برسالة رد، والتي تتضمن دائمًا معلومات رأس وتحتوي عادةً على نص. يبدو السطر الأول من طلب HTTP كما يلي:

```
GET /index.html HTTP/1.1
```

يحدد هذا السطر أمر HTTP، يسمى طريقة "*method*"، متبوعًا بعنوان المستند وإصدار بروتوكول HTTP المستخدم. في هذه الحالة، يستخدم الطلب طريقة GET لطلب مستند index.html باستخدام HTTP 1.1. بعد هذا السطر الأولي، يمكن أن يحتوي الطلب على معلومات رأس اختيارية تمنح الخادم بيانات إضافية حول الطلب.

فمثلا:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

توفر ترويسة **User-Agent** معلومات حول مستعرض الويب، بينما يحدد رأس **Accept** أنواع **MIME** التي يقبلها المستعرض. بعد أي رؤوس، يحتوي الطلب على سطر فارغ للإشارة إلى نهاية قسم الرأس. يمكن أن يحتوي الطلب أيضًا على بيانات إضافية، إذا كان ذلك مناسباً للطريقة المستخدمة (على سبيل المثال، مع طريقة POST، كما سنناقش قريبًا). إذا كان الطلب لا يحتوي على أي بيانات، فإنه ينتهي بسطر فارغ.

يتلقى خادم الويب الطلب ومعالجته وإرسال استجابة. يبدو السطر الأول من استجابة HTTP كما يلي:

```
HTTP/1.1 200 OK
```

يحدد هذا السطر إصدار البروتوكول وكود الحالة ووصفًا لذلك الكود. في هذه الحالة، يكون كود الحالة هو 200، مما يعني أن الطلب كان ناجحًا (ومن ثم الوصف جيد). بعد سطر الحالة، تحتوي الاستجابة على رؤوس تزود العميل بمعلومات إضافية حول الاستجابة. فمثلا:

```
Date: Sat, 29 June 2019 14:07:50 GMT
Server: Apache/2.2.14 (Ubuntu)
Content-Type: text/html
Content-Length: 1845
```

يوفر رأس **Server** معلومات حول برنامج خادم الويب، بينما يحدد رأس **Content-Type** نوع MIME للبيانات المضمنة في الاستجابة. بعد الرؤوس، تحتوي الاستجابة على سطر فارغ، متبوعًا بالبيانات المطلوبة في حالة نجاح الطلب.

أكثر طريقتين HTTP شيوعًا هما GET و POST. تم تصميم طريقة GET لاسترداد المعلومات، مثل مستند أو صورة أو نتائج استعلام قاعدة بيانات من الخادم. طريقة POST مخصصة لنشر المعلومات، مثل: رقم بطاقة الائتمان أو المعلومات التي سيتم تخزينها في قاعدة بيانات، على الخادم. طريقة GET هي ما يستخدمه متصفح الويب عندما يكتب المستخدم عنوان URL أو ينقر على رابط. عندما يرسل المستخدم نموذجًا، يمكن استخدام طريقة GET أو POST، كما هو محدد بواسطة خاصية method لوسم form. سنناقش طرق GET و POST بمزيد من التفصيل في قسم "نماذج المعالجة".

# المتغيرات

تكوين الخادم ومعلومات الطلب - بما في ذلك معلمات النموذج وملفات تعريف الارتباط - يمكن الوصول إليها بثلاث طرق مختلفة من البرامج النصية PHP. بشكل جماعي، يشار إلى هذه المعلومات باسم EGPCS (اختصار لـ البيئة *"environment"*، GET، POST، ملفات تعريف الارتباط *"cookies"*، والخادم *"server"*).

تنشئ PHP ستة مصفوفات عالمية تحتوي على معلومات EGPCS:

***$_ENV***

يحتوي على قيم أي متغيرات بيئة، حيث تكون مفاتيح المصفوفة هي أسماء متغيرات البيئة.

***$_GET***

يحتوي على أي معلمات تشكل جزءًا من طلب GET، حيث تكون مفاتيح المصفوفة هي أسماء معلمات النموذج.

***$_COOKIE***

يحتوي على أي قيم ملفات تعريف ارتباط تم تمريرها كجزء من الطلب، حيث تكون مفاتيح المصفوفة هي أسماء ملفات تعريف الارتباط.

***$_POST***

يحتوي على أي معلمات تشكل جزءًا من طلب POST، حيث تكون مفاتيح المصفوفة هي أسماء معلمات النموذج.

***$_SERVER***

يحتوي على معلومات مفيدة حول خادم الويب، كما هو موضح في القسم التالي.

***$_FILES***

يحتوي على معلومات حول أي ملفات تم تحميلها.

هذه المتغيرات ليست عالمية فقط، ولكنها مرئية أيضًا من داخل تعريفات الدوال. يتم إنشاء المصفوفة $_REQUEST بواسطة PHP تلقائيًا وتحتوي على عناصر المصفوفات $_GET و $_POST و $_COOKIE كلها في متغير مصفوفة واحد.

# معلومات الخادم

تحتوي مصفوفة **$_SERVER** على الكثير من المعلومات المفيدة من خادم الويب، ويأتي الكثير منها من متغيرات البيئة المطلوبة في مواصفات واجهة البوابة المشتركة "**C**ommon **G**ateway **I**nterface" (**CGI**). فيما يلي قائمة كاملة بإدخالات $_SERVER التي تأتي من CGI، بما في ذلك بعض أمثلة القيم:

***PHP_SELF***

اسم البرنامج النصي الحالي، بالنسبة إلى جذر المستند (على سبيل المثال: /store/cart.php). لقد رأيت هذا بالفعل مستخدمًا في بعض نماذج التعليمات البرمجية في الفصول السابقة. هذا المتغير مفيد عند إنشاء نصوص برمجية مرجعية ذاتية، كما سنرى لاحقًا.

***SERVER_SOFTWARE***

سلسلة تحدد الخادم (على سبيل المثال: " Apache/1.3.33 (Unix) mod_perl/1.26 PHP/5.0.4").

***SERVER_NAME***

اسم المضيف أو اسم DNS المستعار أو عنوان IP لعناوين URL المرجعية الذاتية (على سبيل المثال: www.example.com).

***GATEWAY_INTERFACE***

معرفة إصدار معيار CGI (على سبيل المثال: CGI/1.1).

***SERVER_PROTOCOL***

اسم وإصدار بروتوكول الطلب (على سبيل المثال: HTTP/1.1).

***SERVER_PORT***

رقم منفذ الخادم الذي تم إرسال الطلب إليه (على سبيل المثال: 80).

***REQUEST_METHOD***

الطريقة التي استخدمها العميل لجلب المستند (على سبيل المثال: GET).

***PATH_INFO***

عناصر المسارات الإضافية التي قدمها العميل (على سبيل المثال: /list/users).

***PATH_TRANSLATED***

قيمة PATH_INFO، مترجمة من قبل الخادم إلى اسم ملف (على سبيل المثال: /home/httpd/htdocs/list/users).

***SCRIPT_NAME***

مسار URL للصفحة الحالية، والذي يكون مفيدًا للنصوص النصية المرجعية الذاتية (على سبيل المثال: /~me/menu.php).

***QUERY_STRING***

كل شيء بعد؟ في عنوان URL (على سبيل المثال: name=Fred+age=35).

***REMOTE_HOST***

اسم مضيف الجهاز الذي طلب هذه الصفحة (على سبيل المثال، http://dialup-192-168-0-1.example.com). إذا لم يكن هناك DNS للجهاز، فهذا فارغ و REMOTE_ADDR هي المعلومات الوحيدة المقدمة.

***REMOTE_ADDR***

سلسلة تحتوي على عنوان IP للجهاز الذي طلب هذه الصفحة (على سبيل المثال: "192.168.0.250").

***AUTH_TYPE***

طريقة المصادقة المستخدمة لحماية الصفحة، إذا كانت الصفحة محمية بكلمة مرور (على سبيل المثال: basic).

***REMOTE_USER***

اسم المستخدم الذي صادق "authenticated" عليه العميل، إذا كانت الصفحة محمية بكلمة مرور (على سبيل المثال: fred). لاحظ أنه لا توجد طريقة لمعرفة كلمة المرور المستخدمة.

يقوم خادم Apache أيضًا بإنشاء إدخالات في مصفوفة $_SERVER لكل رأس HTTP في الطلب. لكل مفتاح، يتم تحويل اسم الرأس إلى أحرف كبيرة، ويتم تحويل (-) إلى شرطات سفلية (_)، ويتم إضافة السلسلة "HTTP_" مسبقًا. على سبيل المثال، إدخال عنوان User-Agent يحتوي على المفتاح "HTTP_USER_AGENT". العنوانان الأكثر شيوعًا وفائدة هما:

***HTTP_USER_AGENT***

السلسلة التي استخدمها المتصفح لتعريف نفسه (على سبيل المثال: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]").

***HTTP_REFERER***

الصفحة التي قال المتصفح إنها جاءت منها للوصول إلى الصفحة الحالية (على سبيل المثال: http://www.example.com/last_page.html).

# معالجة النماذج

من السهل معالجة النماذج باستخدام PHP، حيث تتوفر معلمات النموذج في مصفوفتي $_GET و $_POST. يصف هذا القسم بعض الحيل والتقنيات التي ستجعل الأمر أكثر سهولة.

## الطرق "Methods"

كما ناقشنا بالفعل، هناك طريقتان HTTP يمكن للعميل استخدامها لتمرير بيانات النموذج إلى الخادم: GET و POST. يتم تحديد الطريقة التي يستخدمها نموذج معين بخاصية method لوسم form. من الناحية النظرية، تعتبر الأساليب غير حساسة لحالة الأحرف في HTML، ولكن في الممارسة العملية، تتطلب بعض المتصفحات المعطلة أن يكون اسم الطريقة بأحرف كبيرة.

يقوم طلب GET بترميز معلمات النموذج في عنوان URL في سلسلة استعلام *"query string"*، والتي يشار إليها بالنص الذي يلي؟:

```
/path/to/chunkify.php?word=despicable&length=3
```

يقوم طلب POST بتمرير معلمات النموذج في نص طلب HTTP، مع ترك عنوان URL دون تغيير.

الفرق الأكثر وضوحًا بين GET و POST هو سطر عنوان URL. نظرًا لأن جميع معلمات النموذج يتم ترميزها في عنوان URL بطلب GET، يمكن للمستخدمين وضع إشارة مرجعية على استعلامات GET. ومع ذلك، لا يمكنهم القيام بذلك مع طلبات POST.

ومع ذلك، فإن الاختلاف الأكبر بين طلبات GET و POST هو أدق بكثير. تنص مواصفات HTTP على أن طلبات GET غير صالحة "*idempotent*" — أي أن طلب GET واحدًا لعنوان URL معين، يتضمن معلمات النموذج، هو نفس طلبين أو أكثر لعنوان URL هذا. وبالتالي، يمكن لمتصفحات الويب تخزين صفحات الاستجابة لطلبات GET مؤقتًا، لأن صفحة الاستجابة لا تتغير بغض النظر عن عدد مرات تحميل الصفحة. بسبب (idempotence)، يجب استخدام طلبات GET فقط للاستعلامات مثل: تقسيم كلمة إلى أجزاء أصغر أو مضاعفة الأرقام، حيث لن تتغير صفحة الاستجابة أبدًا.

طلبات POST ليست معطلة "idempotent". هذا يعني أنه لا يمكن تخزينها مؤقتًا، ويتم الاتصال بالخادم في كل مرة يتم فيها عرض الصفحة. ربما رأيت أن متصفح الويب يطالبك بـ "إعادة نشر بيانات النموذج؟" "Repost form data?" قبل عرض أو إعادة تحميل صفحات معينة. وهذا يجعل طلبات POST الخيار المناسب للاستعلامات التي قد تتغير صفحات استجابتها بمرور الوقت - على سبيل المثال، عرض محتويات عربة التسوق أو الرسائل الحالية في لوحة الإعلانات.

ومع ذلك، غالبًا ما يتم تجاهل الـ(idempotence) في العالم الحقيقي. يتم تنفيذ ذاكرة التخزين المؤقت "caches" للمتصفح بشكل سيئ للغاية، ويسهل الوصول إلى زر إعادة التحميل "reloading"، بحيث يميل المبرمجون إلى استخدام GET و POST بناءً على ما إذا كانوا يريدون معلمات الاستعلام المعروضة في عنوان URL أم لا. ما تحتاج إلى تذكره هو أنه لا ينبغي استخدام طلبات GET لأي إجراءات تسبب تغييرًا في الخادم، مثل تقديم طلب أو تحديث قاعدة بيانات.

يتوفر نوع الطريقة التي تم استخدامها لطلب صفحة PHP من خلال $_SERVER['REQUEST_METHOD']. فمثلا:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
 // handle a GET request
}
else {
 die("You may only GET this page.");
}
```

## المعلمات "Parameters"

استخدم المصفوفات $_POST و $_GET و $_FILES للوصول إلى معلمات النموذج من كود PHP الخاص بك. المفاتيح هي أسماء المعلمات، والقيم هي قيم تلك المعلمات. نظراً لأن النقط "periods" قانونية في أسماء حقول HTML ولكن ليس في أسماء متغيرات PHP، يتم تحويل النقاط في أسماء الحقول إلى شرطات سفلية (_) في المصفوفة.

يوضح المثال 8-1 نموذج HTML يقوم بتقسيم سلسلة يوفرها المستخدم. يحتوي النموذج على حقلين: أحدهما للسلسلة (اسم المعلمة: word) والآخر لحجم القطع "chunks" المطلوب إنتاجها (اسم المعلمة: number).

مثال 8-1. نموذج القطع (chunkify.html)

```
<html>
 <head><title>Chunkify Form</title></head>

 <body>
 <form action="chunkify.php" method="POST">
 Enter a word: <input type="text" name="word" /><br />
```

```
 How long should the chunks be?
 <input type="text" name="number" /><br />
 <input type="submit" value="Chunkify!">
 </form>
</body>

</html>
```

يسرد المثال 8-2 نص PHP برمجي، chunkify.php، الذي يقدم إليه النموذج في المثال 8-1. يقوم البرنامج النصي بنسخ قيم المعلمات إلى متغيرات واستخدامها.

مثال 8-2. البرنامج النصي للقطع (chunkify.php)

```
<?php
$word = $_POST['word'];
$number = $_POST['number'];

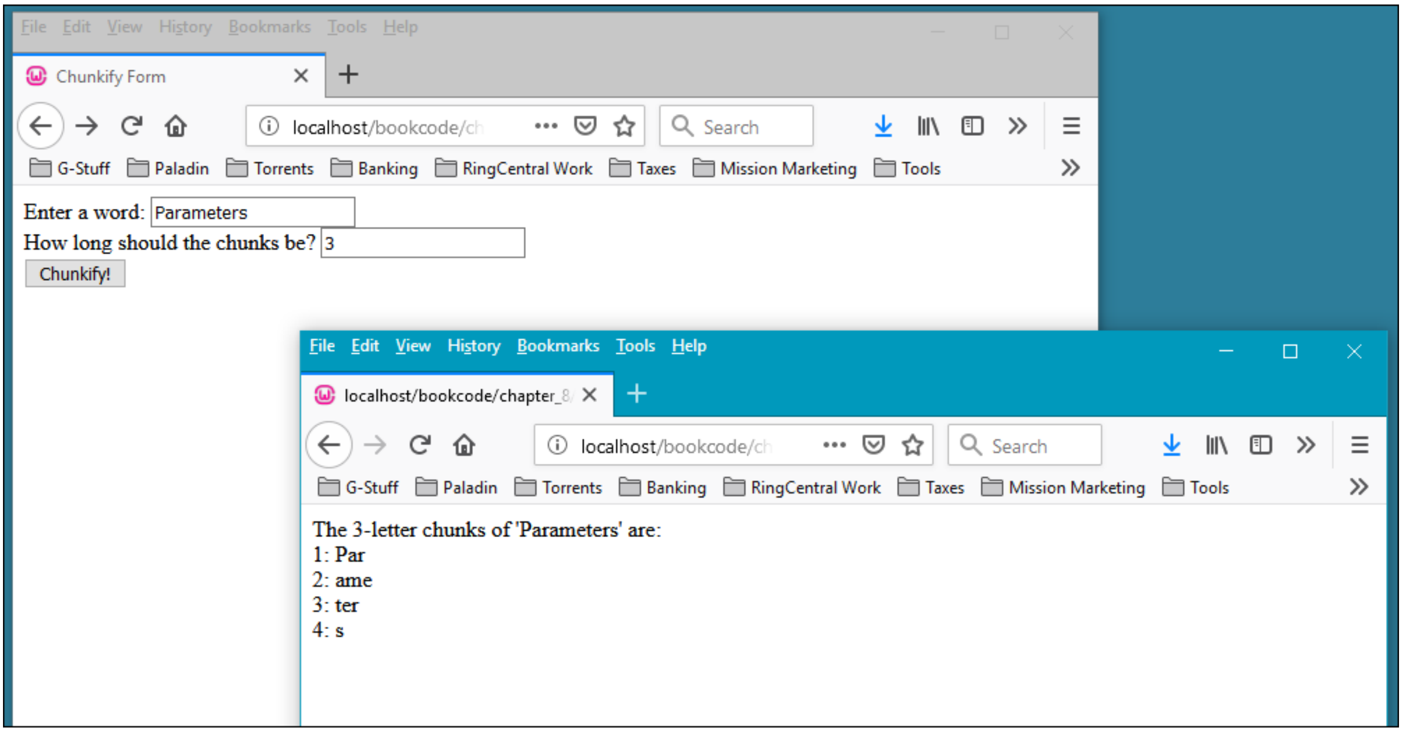
$chunks = ceil(strlen($word) / $number);

echo "The {$number}-letter chunks of '{$word}' are:<br
/>\n";

for ($i = 0; $i < $chunks; $i++) {
 $chunk = substr($word, $i * $number, $number);
 printf("%d: %s<br />\n", $i + 1, $chunk);
```

```
}
?>
```

يوضح الشكل 8-1 كلاً من نموذج chunkify والمخرجات الناتجة.




الشكل 8-1. نموذج chunkify وإخراجه

## صفحات المعالجة الذاتية "Self-Processing Pages"

يمكن استخدام صفحة PHP واحدة لإنشاء نموذج ومعالجته لاحقًا. إذا كانت الصفحة الموضحة في المثال 8-3 مطلوبة باستخدام طريقة GET، فإنها تطبع نموذجًا يقبل درجة حرارة فهرنهايت. ومع ذلك، إذا تم استدعاؤها باستخدام طريقة POST، تحسب الصفحة وتعرض درجة الحرارة المئوية المقابلة.

مثال 8-3. صفحة معالجة ذاتية لتحويل درجة الحرارة (temp.php)

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

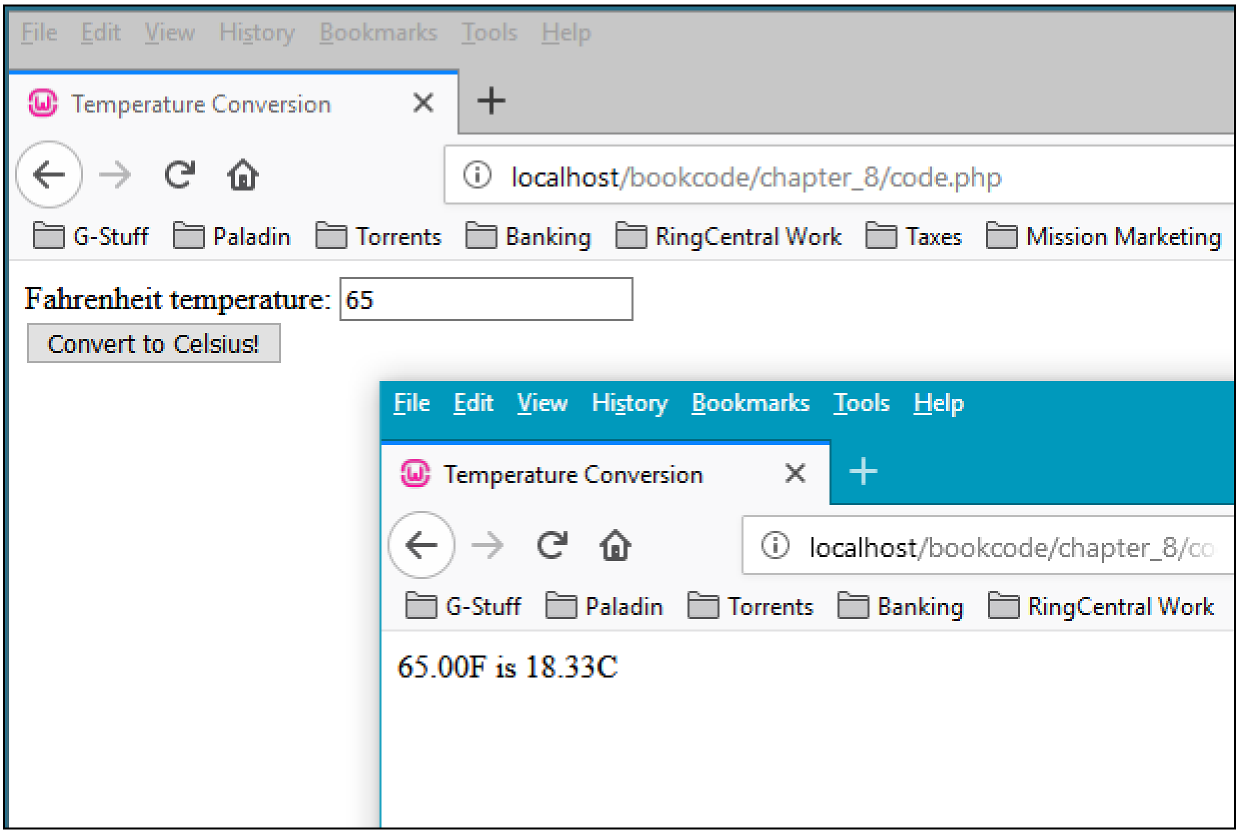
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
 <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
 Fahrenheit temperature:
 <input type="text" name="fahrenheit" /><br />
 <input type="submit" value="Convert to Celsius!" />
 </form>

<?php }
else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
 $fahrenheit = $_POST['fahrenheit'];
 $celsius = ($fahrenheit - 32) * 5 / 9;

 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
}
else {
 die("This script only works with GET and POST requests.");
} ?>

</body>
</html>
```

يوضح الشكل 8-2 صفحة تحويل درجة الحرارة والمخرجات الناتجة.




الشكل 8-2. صفحة تحويل درجة الحرارة وإخراجها

هناك طريقة أخرى للبرنامج النصي ليقرر ما إذا كان سيعرض نموذجًا أو عملية، وهي معرفة ما إذا كان قد تم توفير إحدى المعلمات أم لا. يتيح لك ذلك كتابة صفحة معالجة ذاتية تستخدم طريقة GET لإرسال القيم. يوضح المثال 8-4 نسخة جديدة من صفحة تحويل درجة الحرارة التي ترسل المعلمات باستخدام طلب GET. تستخدم هذه الصفحة وجود أو عدم وجود معلمات لتحديد ما يجب فعله.

مثال 8-4. تحويل درجة الحرارة باستخدام طريقة GET (temp2.php)

```
<html>
<head>
<title>Temperature Conversion</title>
</head>
```

```
<body>
<?php
if (isset ( $_GET ['fahrenheit'] )) {
 $fahrenheit = $_GET ['fahrenheit'];
} else {
 $fahrenheit = null;
}
if (is_null ( $fahrenheit )) {
 ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Fahrenheit temperature: <input type="text"
name="fahrenheit" /><br />
 <input type="submit" value="Convert to Celsius!" />
 </form>
<?php
} else {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf ( "%.2fF is %.2fC", $fahrenheit, $celsius );
}
?>
</body>
</html>
```

في المثال 8-4 ، ننسخ قيمة معلمة النموذج إلى $fahrenheit. إذا لم نعط هذه المعلمة، فإن $fahrenheit يحتوي على NULL، لذلك يمكننا استخدام is_null() لاختبار ما إذا كان ينبغي لنا عرض النموذج أو معالجة بيانات النموذج.

## نماذج لاصقة "Sticky Forms"

تستخدم العديد من مواقع الويب أسلوبًا يُعرف باسم النماذج اللاصقة، حيث تكون نتائج الاستعلام مصحوبة بنموذج بحث تكون قيمه الافتراضية هي قيم الاستعلام السابق. على سبيل المثال، إذا بحثت في Google عن "Programming PHP"، فسيحتوي الجزء العلوي من صفحة النتائج على مربع بحث آخر يحتوي بالفعل على "Programming PHP". لتحسين البحث إلى " Programming PHP from O'Reilly, "، يمكنك ببساطة إضافة الكلمات الرئيسية الإضافية.

هذا السلوك الملصق سهل التنفيذ. يوضح المثال 8-5 البرنامج النصي الخاص بتحويل درجة الحرارة من المثال 8-4، مع جعل النموذج لاصقا. الأسلوب الأساسي هو استخدام قيمة النموذج المقدمة كقيمة افتراضية عند إنشاء حقل HTML.

مثال 8-5. تحويل درجة الحرارة بشكل لاصق (sticky_form.php)

```
<html>
<head><title>Temperature Conversion</title></head>
<body>
<?php $fahrenheit = $_GET['fahrenheit']; ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
```

```
 Fahrenheit temperature:
 <input type="text" name="fahrenheit" value="<?php echo
$fahrenheit; ?>" /><br />
 <input type="submit" value="Convert to Celsius!" />
</form>

<?php if (!is_null($fahrenheit)) {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
} ?>

</body>
</html>
```

### معلمات متعددة القيم "Multivalued Parameters"

يمكن أن تسمح قوائم اختيار HTML، التي تم إنشاؤها باستخدام وسم select، باختيارات متعددة. للتأكد من أن PHP يتعرف على القيم المتعددة التي يمررها المتصفح إلى برنامج نصي لمعالجة النماذج، تحتاج إلى استخدام الأقواس المربعة، []، بعد اسم الحقل في نموذج HTML. فمثلا:

```
<select name="languages[]">

 <option name="c">C</option>
 <option name="c++">C++</option>
 <option name="php">PHP</option>
 <option name="perl">Perl</option>
</select>
```

الآن، عندما يرسل المستخدم النموذج، يحتوي $_GET['languages'] على مصفوفة بدلاً من سلسلة بسيطة. تحتوي هذه المصفوفة على القيم التي تم تحديدها من قبل المستخدم.

يوضح المثال 8-6 تحديدات متعددة للقيم ضمن قائمة اختيار HTML. يوفر النموذج للمستخدم مجموعة من خاصيات شخصية "personality". عندما يرسل المستخدم النموذج، فإنه يعرض وصفًا (ليس مثيرًا للاهتمام) لشخصية المستخدم.

مثال 8-6. قيم اختيار متعددة مع مربع التحديد (select_array.php)

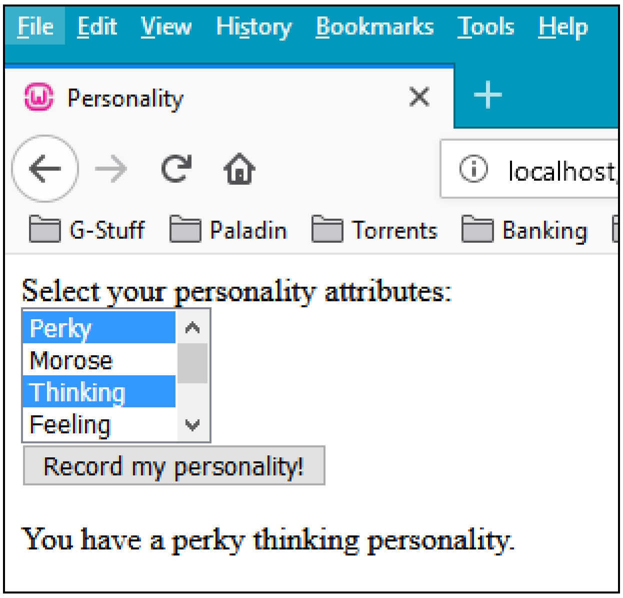
```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Select your personality attributes:<br />
 <select name="attributes[]" multiple>
 <option value="perky">Perky</option>
 <option value="morose">Morose</option>
 <option value="thinking">Thinking</option>
 <option value="feeling">Feeling</option>
 <option value="thrifty">Spend-thrift</option>
 <option value="shopper">Shopper</option>
 </select><br />
 <input type="submit" name="s" value="Record my
personality!" />
```

```
</form>
<?php if (array_key_exists('s', $_GET)) {
 $description = join(' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

في المثال 8-6، زر الإرسال له اسم، "s". تحقق من وجود قيمة المعلمة هذه لمعرفة ما إذا كان يوجد وصف للشخصية. يوضح الشكل 8-3 صفحة التحديد المتعدد والمخرجات الناتجة.




الشكل 8-3. صفحة متعددة التحديد وإخراجها

تنطبق نفس التقنية على أي حقل نموذج حيث يمكن إرجاع قيم متعددة. يوضح المثال 8-7 نسخة منقحة من نموذج شخصيتنا الذي تمت إعادة كتابته لاستخدام مربعات الاختيار بدلاً من مربع التحديد. لاحظ

أنه تم تغيير HTML فقط - لا تحتاج التعليمات البرمجية لمعالجة النموذج إلى معرفة ما إذا كانت القيم المتعددة تأتي من مربعات الاختيار "checkboxes" أم من مربع الاختيار "select box".

مثال 8-7. قيم اختيار متعددة في مربعات الاختيار (checkbox_array.php)

```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php $_SERVER['PHP_SELF']; ?>"
method="GET">
 Select your personality attributes:<br />
 <input type="checkbox" name="attributes[]"
value="perky" /> Perky<br />
 <input type="checkbox" name="attributes[]"
value="morose" /> Morose<br />
 <input type="checkbox" name="attributes[]"
value="thinking" /> Thinking<br />
 <input type="checkbox" name="attributes[]"
value="feeling" /> Feeling<br />
 <input type="checkbox" name="attributes[]"
value="thrifty" />Spend-thrift<br />
 <input type="checkbox" name="attributes[]"
value="shopper" /> Shopper<br />
 <br />
 <input type="submit" name="s" value="Record my
personality!" />
</form>
<?php if (array_key_exists('s', $_GET)) {
```

```
 $description = join (' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

## معلمات لاصقة متعددة القيم "Sticky Multivalued Parameters"

لذا ربما تتساءل الآن، هل يمكنني جعل عناصر النموذج ذات التحديد المتعدد لاصقة؟ يمكنك ذلك، لكن هذا ليس بالأمر السهل. ستحتاج إلى التحقق مما إذا كانت كل قيمة ممكنة في النموذج إحدى القيم المرسلة. فمثلا:

```
Perky: <input type="checkbox" name="attributes[]" value="perky"
<?php
if (is_array($_GET['attributes']) && in_array('perky', $_GET['attributes'])) {
 echo "checked";
} ?> /><br />
```

يمكنك استخدام هذه التقنية لكل مربع اختيار "checkbox"، ولكن هذا متكرر وعرضة للخطأ "error-prone". في هذه المرحلة، يكون من الأسهل كتابة دالة لإنشاء HTML للقيم المحتملة والعمل من نسخة من المعلمات المقدمة. يوضح المثال 8-8 نسخة جديدة من مربعات الاختيار "checkboxes" متعددة التحديد، مع جعل النموذج لاصقا. على الرغم من أن هذا النموذج يشبه تمامًا النموذج الموجود في المثال 8-7، إلا أنه توجد خلف الكواليس تغييرات جوهرية في طريقة إنشاء النموذج.

مثال 8-8. مربعات الاختيار اللاصقة متعددة القيم (checkbox_array2.php)

```
<html>
<head><title>Personality</title></head>
<body>
<?php // fetch form values, if any
$attrs = $_GET['attributes'];

if (!is_array($attrs)) {
 $attrs = array();
}

// create HTML for identically named checkboxes

function makeCheckboxes($name, $query, $options)
{
 foreach ($options as $value => $label) {
 $checked = in_array($value, $query) ? "checked" : '';

 echo "<input type=\"checkbox\" name=\"{$name}\"
 value=\"{$value}\" {$checked} />";
 echo "{$label}<br />\n";
 }
}

// the list of values and labels for the checkboxes
```

```
$personalityAttributes = array(
 'perky' => "Perky",
 'morose' => "Morose",
 'thinking' => "Thinking",
 'feeling' => "Feeling",
 'thrifty' => "Spend-thrift",
 'prodigal' => "Shopper"
); ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Select your personality attributes:<br />
 <?php makeCheckboxes('attributes[]', $attrs, $personalityAttributes); ?><br />

 <input type="submit" name="s" value="Record my personality!" />
</form>

<?php if (array_key_exists('s', $_GET)) {
 $description = join (' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

قلب هذا الكود هو دالة ()makeCheckboxes. يتطلب الأمر ثلاث مدخلات: اسم مجموعة مربعات الاختيار، ومجموعة القيم الافتراضية، والمصفوفة التي تعين القيم للأوصاف "descriptions". توجد قائمة خيارات مربعات الاختيار في المصفوفة personalityAttributes$.

## تحميلات الملف "File Uploads"

للتعامل مع تحميلات الملفات (المدعومة في معظم المتصفحات الحديثة)، استخدم مصفوفة FILES_$. باستخدام وظائف المصادقة المختلفة وتحميل الملفات، يمكنك التحكم في من يُسمح له بتحميل الملفات وماذا تفعل بهذه الملفات بمجرد أن تكون على نظامك. تم وصف المخاوف الأمنية التي يجب أخذها في الاعتبار في الفصل 14.

يعرض الكود التالي نموذجًا يسمح بتحميل الملفات إلى نفس الصفحة:

```
<form enctype="multipart/form-data"
 action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 <input type="hidden" name="MAX_FILE_SIZE" value="10240">
 File name: <input name="toProcess" type="file" />
 <input type="submit" value="Upload" />
</form>
```

أكبر مشكلة في عمليات تحميل الملفات هي مخاطر الحصول على ملف كبير جدًا بحيث لا يمكن معالجته. PHP لديها طريقتان لمنع ذلك: حد صارم "hard limit" وحد مرن "soft limit". يوفر خيار upload_max_filesize في php.ini حدًا أقصى لحجم الملفات المرفوعة (يتم تعيينه افتراضياً على 2

ميغابايت). إذا أرسل النموذج الخاص بك معلمة تسمى MAX_FILE_SIZE قبل أي معلمات لحقل الملف، فإن PHP تستخدم هذه القيمة على أنها الحد الأعلى المرن. على سبيل المثال، في المثال السابق، تم تعيين الحد الأعلى على 10 كيلوبايت. تتجاهل PHP محاولات تعيين MAX_FILE_SIZE على قيمة أكبر من upload_max_filesize.

لاحظ أيضًا أن وسم form يأخذ خاصية enctype بالقيمة "multipart/form-data".

كل عنصر في $_FILES هو في حد ذاته مصفوفة تعطي معلومات عن الملف الذي تم تحميله. المفاتيح هي:

الاسم ***"name"***

اسم الملف الذي تم تحميله كما تم توفيره بواسطة المتصفح. من الصعب الاستفادة من ذلك بشكل هادف، حيث قد يكون لجهاز العميل اصطلاحات اسم ملف مختلفة عن خادم الويب (على سبيل المثال، مسار ملف *D:\PHOTOS\ME.JPG* من جهاز عميل يعمل بنظام Windows سيكون بلا معنى لخادم الويب المشغل بيونكس).

النوع ***"type"***

نوع MIME للملف الذي تم تحميله كما توقعه العميل.

**الحجم "size"**

حجم الملف الذي تم تحميله (بالبايت). إذا حاول المستخدم تحميل ملف كبير جدًا، فسيتم الإبلاغ عن الحجم على أنه 0.

**الاسم المؤقت "tmp_name"**

اسم الملف المؤقت "temporary file" على الخادم الذي يحتفظ بالملف الذي تم تحميله. إذا حاول المستخدم تحميل ملف كبير جدًا، فسيتم إعطاء الاسم كـ "none".

الطريقة الصحيحة لاختبار ما إذا تم تحميل ملف بنجاح هي استخدام الدالة is_uploaded_file()، على النحو التالي:

```
if
(is_uploaded_file($_FILES['toProcess']['tmp_name'])) {
 // successfully uploaded
}
```

يتم تخزين الملفات في دليل الملفات المؤقتة الافتراضي للخادم، المحدد في ملف php.ini باستخدام الخيار upload_tmp_dir. لنقل ملف، استخدم دالة move_uploaded_file():

```
move_uploaded_file($_FILES['toProcess']['tmp_name'],
"path/to/put/file/{$file}");
```

يقوم استدعاء move_uploaded_file() بالتحقق تلقائيًا مما إذا كان ملفًا تم تحميله. عند انتهاء البرنامج النصي، يتم حذف أي ملفات تم تحميلها إلى هذا البرنامج النصي من الدليل المؤقت.

## التحقق من صحة النموذج "Form Validation"

عندما تسمح للمستخدمين بإدخال البيانات، فإنك تحتاج عادةً إلى التحقق من صحة تلك البيانات قبل استخدامها أو تخزينها لاستخدامها لاحقًا. هناك العديد من الاستراتيجيات المتاحة للتحقق من صحة البيانات. الأول هو JavaScript من جانب العميل. ومع ذلك، نظراً لأن المستخدم يمكنه اختيار إيقاف تشغيل JavaScript، أو ربما يستخدم متصفحًا لا يدعمه، فلا يمكن أن يكون هذا هو التحقق الوحيد الذي تقوم به.

الخيار الأكثر أمانًا هو استخدام PHP لإجراء التحقق من الصحة. يوضح المثال 8-9 صفحة معالجة ذاتية مع نموذج. تسمح الصفحة للمستخدم بإدخال عنصر وسائط "media item"؛ ثلاثة من عناصر النموذج - الاسم ونوع الوسائط واسم الملف - مطلوبة. إذا أهمل المستخدم إعطاء قيمة لأي منها، فسيتم تقديم الصفحة من جديد برسالة توضح الخطأ بالتحديد. يتم تعيين أي حقول نموذج قام المستخدم بملئها بالفعل على القيم التي تم إدخالها في الأصل. أخيرًا، كدليل إضافي للمستخدم، يتغير نص زر الإرسال من "Create" إلى "Continue" عندما يقوم المستخدم بتصحيح النموذج.

مثال 8-9. التحقق من صحة النموذج (data_validation.php)

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];

$tried = ($_POST['tried'] == 'yes');
```

```
if ($tried) {
 $validated = (!empty($name) && !empty($mediaType) &&
!empty($filename));

 if (!$validated) { ?>
 <p>The name, media type, and filename are required
fields. Please fill
 them out to continue.</p>
 <?php }
}

if ($tried && $validated) {
 echo "<p>The item has been created.</p>";
}

// was this type of media selected? print "selected" if
so
function mediaSelected($type)
{
 global $mediaType;

 if ($mediaType == $type) {
 echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
```

```
 Name: <input type="text" name="name" value="<?php echo
$name; ?>" /><br />


 Status: <input type="checkbox" name="status"
value="active"
 <?php if ($status == "active") { echo "checked"; } ?>
/> Active<br />


 Media: <select name="media_type">
 <option value="">Choose one</option>
 <option value="picture" <?php
mediaSelected("picture"); ?> />Picture</option>
 <option value="audio" <?php mediaSelected("audio"); ?>
/>Audio</option>
 <option value="movie" <?php mediaSelected("movie"); ?>
/>Movie</option>
 </select><br />


 File: <input type="text" name="filename" value="<?php
echo $filename; ?>" /><br />


 Caption: <textarea name="caption"><?php echo $caption;
?></textarea><br />


 <input type="hidden" name="tried" value="yes" />
 <input type="submit" value="<?php echo $tried ?
"Continue" : "Create"; ?>" />
</form>
```

في هذه الحالة، يكون التحقق من الصحة مجرد التحقق من تقديم قيمة. قمنا بتعيين التحقق من الصحة $validated ليكون true فقط إذا كانت $name و $type و $filename كلها غير فارغة. تتضمن عمليات التحقق المحتملة الأخرى التحقق من صحة عنوان البريد الإلكتروني أو التحقق من أن اسم الملف المقدم محلي وموجود.

على سبيل المثال، للتحقق من صحة حقل العمر "age" للتأكد من احتوائه على عدد صحيح غير سالب، استخدم هذا الكود:

```
$age = $_POST['age'];
$validAge = strspn($age, "1234567890") == strlen($age);
```

يعثر استدعاء ()strspn على عدد الأرقام في بداية السلسلة. في الأعداد الصحيحة غير السالبة، يجب أن تتكون السلسلة بأكملها من أرقام، لذا فهو عمر صالح إذا كانت السلسلة بأكملها مكونة من أرقام. كان بإمكاننا أيضًا إجراء هذا التحقق باستخدام تعبير عادي:

```
$validAge = preg_match('/^\d+$/', $age);
```

التحقق من صحة عناوين البريد الإلكتروني مهمة قريبة من المستحيل. لا توجد طريقة لأخذ سلسلة ومعرفة ما إذا كانت تتوافق مع عنوان بريد إلكتروني صالح. ومع ذلك، يمكنك اكتشاف الأخطاء الإملائية من خلال مطالبة المستخدم بإدخال عنوان البريد الإلكتروني مرتين (في حقلين مختلفين). يمكنك أيضًا منع الأشخاص من إدخال عناوين البريد الإلكتروني مثل: *me* أو *me@aol* عن طريق طلب علامة (@) ونقطة في مكان ما بعد ذلك، وللحصول على نقاط المكافأة، يمكنك التحقق من المجالات التي لا تريد إرسال بريد إليها ( على سبيل المثال، whitehouse.gov، أو موقع منافس). فمثلا:

```
$email1 = strtolower($_POST['email1']);
$email2 = strtolower($_POST['email2']);
```

```php
if ($email1 !== $email2) {
 die("The email addresses didn't match");
}

if (!preg_match('/@.+\..+$/', $email1)) {
 die("The email address is malformed");
}

if (strpos($email1, "whitehouse.gov")) {
 die("I will not send mail to the White House");
}
```

التحقق من الحقول هو في الأساس معالجة السلسلة. في هذا المثال، استخدمنا التعبيرات العادية ودوال السلاسل للتأكد من أن السلسلة التي يقدمها المستخدم هي نوع السلسلة التي نتوقعها.

# تعيين رؤوس الاستجابة

**Setting Response Headers**

### الى هنا انتهت الترجمة ... باقي المستند فيه عن الفصل التاسع (ترجمت منه لأنه كان عندي امتحان فيه ^^) ##

....

....

...

...

**ننتقل للفصل التاسع**

# الفصل التاسع: قواعد البيانات

تدعم PHP أكثر من 20 قاعدة بيانات، بما في ذلك الأنواع التجارية ومفتوحة المصدر الأكثر شيوعًا. تعد أنظمة قواعد البيانات الارتباطية مثل MariaDB و MySQL و PostgreSQL و Oracle العمود الفقري لمعظم مواقع الويب الحيوية الحديثة. في هذه يتم تخزين معلومات عربة التسوق، وتاريخ الشراء، ومراجعات المنتجات، ومعلومات المستخدم، وأرقام بطاقات الائتمان، وأحيانًا صفحات الويب نفسها.

يتناول هذا الفصل كيفية الوصول إلى قواعد البيانات من PHP. نحن نركز على مكتبة كائنات بيانات *"PHP Data Objects"* (PDO) PHP المدمجة، والتي تتيح لك استخدام نفس الدوال للوصول إلى أي قاعدة بيانات، بدلاً من الامتدادات التي لا تعد ولا تحصى الخاصة بقاعدة البيانات. في هذا الفصل، ستتعلم كيفية جلب البيانات من قاعدة البيانات، وتخزين البيانات في قاعدة البيانات، والتعامل مع الأخطاء. ننتهي من نموذج تطبيق يوضح كيفية تنفيذ تقنيات قواعد البيانات المختلفة.

لا يمكن لهذا الكتاب الخوض في جميع تفاصيل إنشاء تطبيقات قواعد بيانات الويب باستخدام PHP. لإلقاء نظرة أكثر تعمقًا على مجموعة PHP/MySQL، راجع *Web Database Applications with PHP and MySQL*, Second Edition (O'Reilly)، بواسطة Hugh Williams و David Lane.

# استخدام PHP للوصول إلى قاعدة بيانات

هناك طريقتان للوصول إلى قواعد البيانات من PHP. الأول هو استخدام ملحق خاص بقاعدة البيانات، والآخر هو استخدام مكتبة PDO المستقلة عن قاعدة البيانات. هناك مزايا وعيوب لكل طريقة.

إذا كنت تستخدم امتدادًا خاصًا بقاعدة البيانات، فإن كودك مرتبط ارتباطًا وثيقًا بقاعدة البيانات التي تستخدمها. على سبيل المثال، تختلف أسماء دوال امتداد MySQL والمعلمات ومعالجة الأخطاء وما إلى ذلك تمامًا عن تلك الخاصة بامتدادات قاعدة البيانات الأخرى. إذا كنت ترغب في نقل قاعدة البيانات الخاصة بك من MySQL إلى PostgreSQL، فستتضمن تغييرات كبيرة في التعليمات البرمجية الخاصة بك. من ناحية أخرى، تخفي PDO الدوال الخاصة بقاعدة البيانات عنك بطبقة تجريدية، لذلك يمكن أن يكون التنقل بين أنظمة قاعدة البيانات بسيطًا مثل تغيير سطر واحد من برنامجك أو ملف php.ini.

تأتي قابلية نقل طبقة التجريد مثل مكتبة PDO بثمن، حيث أن الكود الذي يستخدمها يكون عادةً أبطأ قليلاً من الكود الذي يستخدم امتدادًا أصليًا خاصًا بقاعدة البيانات.

ضع في اعتبارك أن طبقة التجريد لا تفعل شيئًا على الإطلاق عندما يتعلق الأمر بالتأكد من أن استعلامات SQL الفعلية الخاصة بك قابلة للنقل. إذا كان تطبيقك يستخدم أي نوع من أنواع SQL غير العامة، فسيتعين عليك القيام بعمل مهم لتحويل استعلاماتك من قاعدة بيانات إلى أخرى. سننظر بإيجاز في كلا الطريقتين المتبعين في واجهات قواعد البيانات في هذا الفصل، ثم ننظر في الطرق البديلة لإدارة المحتوى الحيوي للويب.

# قواعد البيانات العلائقية و SQL

**Relational Databases and SQL**

نظام إدارة قواعد البيانات العلائقية "Relational Database Management System" (RDBMS) هو خادم يدير البيانات نيابة عنك. يتم تنظيم البيانات في جداول، حيث يحتوي كل جدول على عدد من الأعمدة، ولكل منها اسم ونوع. على سبيل المثال، لتتبع كتب الخيال العلمي، قد يكون لدينا جدول "كتب" يسجل العنوان (سلسلة) وسنة الإصدار (رقم) والمؤلف.

يتم تجميع الجداول معًا في قواعد بيانات، لذلك قد تحتوي قاعدة بيانات كتب الخيال العلمي على جداول لفترات زمنية ومؤلفين وأشرار. عادةً ما يكون لنظام RDBMS نظام مستخدم خاص به ، والذي يتحكم في حقوق الوصول لقواعد البيانات (على سبيل المثال ، "يمكن للمستخدم فريد تحديث مؤلفي قاعدة البيانات").

تتواصل PHP مع قواعد البيانات العلائقية مثل MariaDB و Oracle باستخدام لغة الاستعلام الهيكلية (SQL). يمكنك استخدام SQL لإنشاء قواعد البيانات العلائقية وتعديلها والاستعلام عنها.

ينقسم بناء جملة SQL إلى جزأين. تُستخدم اللغة الأولى ، وهي لغة معالجة البيانات (DML) ، لاسترداد البيانات وتعديلها في قاعدة بيانات موجودة. DML مضغوط بشكل ملحوظ ، ويتألف من أربعة إجراءات أو أفعال فقط: SELECT و INSERT و UPDATE و DELETE. تُعرف مجموعة أوامر SQL المستخدمة لإنشاء هياكل قواعد البيانات التي تحتوي على البيانات وتعديلها باسم لغة تعريف البيانات أو DDL. بناء جملة DDL ليس موحدًا مثل ذلك الخاص بـ DML ، ولكن نظرًا لأن PHP ترسل فقط

أي أوامر SQL تعطيها إلى قاعدة البيانات ، يمكنك استخدام أي أوامر SQL تدعمها قاعدة البيانات الخاصة بك.

ملحوظة

يتوفر ملف أوامر SQL لإنشاء نموذج قاعدة بيانات المكتبة في ملف يسمى library.sql.

بافتراض أن لديك جدولًا يسمى الكتب ، فإن عبارة SQL هذه ستدرج صفًا جديدًا:

**INSERT INTO** books **VALUES** (**null**, 4, 'I, Robot', '0-553-29438-5', 1950, 1);

تُدرج عبارة SQL هذه صفًا جديدًا ولكنها تحدد الأعمدة التي توجد لها قيم:

**INSERT INTO** books (authorid, title, ISBN, pub_year, available)

**VALUES** (4, 'I, Robot', '0-553-29438-5', 1950, 1);

لحذف جميع الكتب التي تم نشرها عام 1979 (إن وجدت) ، يمكننا استخدام عبارة SQL هذه:

**DELETE FROM** books **WHERE** pub_year = 1979;

لتغيير سنة Roots إلى 1983 ، استخدم عبارة SQL هذه:

**UPDATE** books **SET** pub_year=1983 **WHERE** title='Roots';

لجلب الكتب المنشورة في الثمانينيات فقط ، استخدم:

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

يمكنك أيضًا تحديد الحقول التي تريد إرجاعها. فمثلا:

```
SELECT title, pub_year FROM books WHERE pub_year > 1979 AND pub_year
< 1990;
```

يمكنك إصدار استعلامات تجمع المعلومات من جداول متعددة. على سبيل المثال ، يجمع هذا الاستعلام بين جدولي الكتّاب والمؤلفين للسماح لنا بمعرفة من كتب كل كتاب:

```
SELECT authors.name, books.title FROM books, authors
 WHERE authors.authorid = books.authorid;
```

يمكنك حتى إنشاء شكل قصير (أو اسم مستعار) لأسماء الجداول كما يلي:

```
SELECT a.name, b.title FROM books b, authors a WHERE a.authorid = b.authorid;
```

لمزيد من المعلومات حول SQL ، راجع SQL في موجز ، الإصدار الثالث (O'Reilly) ، بقلم كيفن كلاين.

كائنات بيانات PHP

هذا ما يقوله موقع PHP عن PDO:

يحدد امتداد PHP Data Objects (PDO) واجهة خفيفة الوزن ومتسقة للوصول إلى قواعد البيانات في PHP. يمكن لكل برنامج تشغيل قاعدة بيانات ينفذ واجهة PDO الكشف عن ميزات خاصة بقاعدة البيانات كوظائف امتداد عادية. لاحظ أنه لا يمكنك أداء أي وظائف قاعدة بيانات باستخدام ملحق PDO بمفرده ، يجب عليك استخدام برنامج تشغيل PDO خاص بقاعدة البيانات للوصول إلى خادم قاعدة البيانات.

من بين ميزاته الفريدة الأخرى ، PDO:

هو امتداد C أصلي

يستفيد من أحدث مكونات PHP 7 الداخلية

يستخدم القراءة المخزنة للبيانات من مجموعة النتائج

يوفر ميزات قاعدة البيانات المشتركة كقاعدة

لا يزال قادرًا على الوصول إلى الوظائف الخاصة بقاعدة البيانات

يمكن استخدام التقنيات القائمة على المعاملات

يمكن أن تتفاعل مع LOBS (الكائنات الكبيرة) في قاعدة البيانات

يمكن استخدام جمل SQL المعدة والقابلة للتنفيذ مع معلمات منضم

يمكن تنفيذ مؤشرات التمرير

لديه حق الوصول إلى رموز خطأ SQLSTATE ولديه معالجة الأخطاء مرنة للغاية

نظرًا لوجود عدد من الميزات هنا ، سنتطرق إلى القليل منها فقط لتوضيح مدى فائدة PDO.

أولاً ، قليلاً عن PDO. يحتوي على برامج تشغيل لجميع محركات قواعد البيانات الموجودة تقريبًا ، ويجب الوصول إلى تلك المحركات التي لا توفرها PDO من خلال اتصال ODBC العام الخاص بـ PDO. PDO هو نظام معياري من حيث أنه يجب أن يكون لديه على الأقل امتدادين ممّكنين ليكونا نشطين: امتداد PDO نفسه وامتداد PDO الخاص بقاعدة البيانات التي ستتفاعل معها. راجع الوثائق عبر الإنترنت) لإعداد الاتصالات لقاعدة البيانات التي تختارها. كمثال ، لإنشاء PDO على خادم Windows لتفاعل MySQL ، أدخل السطرين التاليين في ملف php.ini وأعد تشغيل الخادم الخاص بك:

```
extension=php_pdo.dll

extension=php_pdo_mysql.dll
```

مكتبة PDO هي أيضًا امتداد موجه للكائنات (كما سترى في أمثلة التعليمات البرمجية التالية).

إجراء اتصال

الشرط الأول لـ PDO هو إجراء اتصال بقاعدة البيانات المعنية والاحتفاظ بهذا الاتصال في متغير مقبض الاتصال ، كما في الكود التالي:

```php
$db = new PDO($dsn, $username, $password);
```

يرمز $ dsn إلى اسم مصدر البيانات ، والمعلمتان الأخريان تشرحان نفسها بنفسها. على وجه التحديد ، بالنسبة لاتصال MySQL ، يمكنك كتابة الكود التالي:

```php
$db = new PDO("mysql:host=localhost;dbname=library", "petermac", "abc123");
```

بالطبع ، يمكنك (يجب) الاحتفاظ بمعلمات اسم المستخدم وكلمة المرور المتغيرة لإعادة استخدام الكود ولأسباب تتعلق بالمرونة.

التعامل مع قاعدة البيانات

بمجرد الاتصال بمحرك قاعدة البيانات وقاعدة البيانات التي تريد التفاعل معها ، يمكنك استخدام هذا الاتصال لإرسال أوامر SQL إلى الخادم. قد تبدو عبارة UPDATE البسيطة كما يلي:

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

يقوم هذا الرمز ببساطة بتحديث جدول الكتب وإصدار الاستعلام. يتيح لك هذا إرسال أوامر SQL بسيطة (على سبيل المثال ، UPDATE، DELETE، INSERT) مباشرة إلى قاعدة البيانات.

استخدام PDO والبيانات المعدة

بشكل نموذجي ، ستستخدم كشوف الحسابات المعدة ، وإصدار مكالمات PDO على مراحل أو خطوات. ضع في اعتبارك الكود التالي:

```
$statement = $db->prepare("SELECT * FROM books");

$statement->execute();


// handle row results, one at a time

while($row = $statement->fetch()) {

 print_r($row);
```

```
// ... or probably do something more meaningful with each returned row

}


$statement = null;
```

في هذه الشفرة ، "نجهز" شفرة SQL ثم "ننفذها". بعد ذلك ، ننتقل عبر النتيجة باستخدام الكود while ، وأخيرًا ، نحرر الكائن الناتج عن طريق تعيين قيمة null له. قد لا يبدو هذا بهذه القوة في هذا المثال البسيط ، ولكن هناك ميزات أخرى يمكن استخدامها مع البيانات المعدة. الآن ، ضع في اعتبارك هذا الرمز:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (:authorid, :title, :ISBN, :pub_year)");


$statement->execute(array(
 'authorid' => 4,
 'title' => "Foundation",
 'ISBN' => "0-553-80371-9",
 'pub_year' => 1951),
);
```

هنا ، نقوم بإعداد بيان SQL بأربعة عناصر نائبة مسماة: المؤلف ، العنوان ، ISBN ، و pub_year. في هذه الحالة ، تصادف أن تكون هذه هي نفس أسماء الأعمدة في قاعدة البيانات ، ولكن يتم ذلك للتوضيح فقط - يمكن أن تكون أسماء العناصر النائبة أي شيء له معنى بالنسبة لك. في استدعاء التنفيذ ، نستبدل هذه العناصر النائبة بالبيانات الفعلية التي نريد استخدامها في هذا الاستعلام المحدد. تتمثل إحدى مزايا العبارات المعدة في أنه يمكنك تنفيذ نفس أمر SQL وتمرير قيم مختلفة عبر المصفوفة في كل مرة. يمكنك أيضًا القيام بهذا النوع من إعداد البيان باستخدام العناصر النائبة الموضعية (وليس تسميتها فعليًا) ، والتي يُشار إليها بـ؟ ، وهو العنصر الموضعي الذي سيتم استبداله. انظر إلى الشكل التالي من الكود السابق:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (?, ?, ?, ?)");


$statement->execute(array(4, "Foundation", "0-553-80371-9", 1951));
```

هذا يحقق نفس الشيء ولكن برمز أقل ، حيث أن منطقة القيمة في عبارة SQL لا تحدد العناصر المراد استبدالها ، وبالتالي فإن المصفوفة في جملة التنفيذ تحتاج إلى إرسال البيانات الأولية فقط دون أسماء. عليك فقط التأكد من موضع البيانات التي ترسلها في البيان المعد.

التعامل مع المعاملات

تدعم بعض أنظمة إدارة قواعد البيانات (RDBMS) المعاملات ، حيث يمكن تنفيذ سلسلة من تغييرات قاعدة البيانات (جميعها مطبقة مرة واحدة) أو التراجع عنها (تم إهمالها ، مع عدم تطبيق أي من التغييرات على قاعدة البيانات). على سبيل المثال ، عندما يتعامل بنك مع تحويل أموال ، يجب أن يتم السحب من

حساب والإيداع في حساب آخر معًا - ولا يجب أن يحدث أي منهما دون الآخر ، ويجب ألا يكون هناك فارق زمني بين الإجراءين. تتعامل PDO مع المعاملات بأناقة باستخدام هياكل try ... catch مثل هذه في المثال 9-1.

مثال 9-1. حاول ... catch بنية الكود

```
try {
 // connection successful
 $db = new PDO("mysql:host=localhost;dbname=banking_sys", "petermac",
"abc123");
} catch (Exception $error) {
 die("Connection failed: " . $error->getMessage());
}


try {
 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $db->beginTransaction();


 $db->exec("insert into accounts (account_id, amount) values (23, '5000')" );
 $db->exec("insert into accounts (account_id, amount) values (27, '-5000')" );
```

```
$db->commit();
} catch (Exception $error) {
$db->rollback();
echo "Transaction not completed: " . $error->getMessage();
}
```

إذا تعذر إكمال المعاملة بالكامل ، فلن يتم إكمال أي منها ، وسيتم طرح استثناء.

إذا قمت باستدعاء الالتزام () أو التراجع () على قاعدة بيانات لا تدعم المعاملات ، فإن الطرق ترجع DB_ERROR.

ملحوظة

تأكد من التحقق من منتج قاعدة البيانات الأساسي الخاص بك للتأكد من أنه يدعم المعاملات.

تصديق البيانات

توفر واجهة PDO طريقة لعرض تفاصيل حول عبارة PDO ، والتي يمكن أن تكون مفيدة لتصحيح الأخطاء إذا حدث خطأ ما.

```
$statement = $db->prepare("SELECT title FROM books WHERE authorid = ?)";


$statement->bindParam(1, "12345678", PDO::PARAM_STR);

$statement->execute();


$statement->debugDumpParams();
```

يؤدي استدعاء طريقة debugDumpParams () على كائن البيان إلى طباعة مجموعة متنوعة من المعلومات حول الاستدعاء:

```
SQL: [35] SELECT title

 FROM books

 WHERE authorID = ?

Sent SQL: [44] SELECT title

 FROM books

 WHERE authorid = "12345678"

Params: 1

Key: Position #0:

paramno=0

name[0] ""
```

is_param=1

param_type=2

يتم عرض قسم إرسال SQL فقط بعد تنفيذ العبارة ؛ قبل ذلك ، يتوفر فقط أقسام SQL و Params.

واجهة كائن MySQLi

منصة قواعد البيانات الأكثر شيوعًا المستخدمة مع PHP هي قاعدة بيانات MySQL. إذا نظرت إلى موقع MySQL على الويب ، فستكتشف أن هناك عددًا قليلاً من الإصدارات المختلفة من MySQL التي يمكنك استخدامها. سننظر في الإصدار القابل للتوزيع المجاني والمعروف باسم خادم المجتمع. تحتوي PHP أيضًا على عدد من الواجهات المختلفة لأداة قاعدة البيانات هذه ، لذلك سننظر في الواجهة الموجهة للكائنات المعروفة باسم MySQLi ، والمعروفة أيضًا بالامتداد المحسن لـ MySQL.

في الآونة الأخيرة ، بدأت MariaDB في تجاوز MySQL كقاعدة بيانات مفضلة لمطوري PHP. حسب التصميم ، فإن MariaDB هي لغة العميل - وأداة الاتصال - والملف الثنائي - متوافقة مع MySQL ؛ هذا يعني أنه يمكنك تثبيت MariaDB وإلغاء تثبيت MySQL وتوجيه تهيئة PHP إلى MariaDB بدلاً من ذلك ، ومن المحتمل ألا تحتاج إلى تغييرات أخرى.

إذا لم تكن معتادًا جدًا على واجهات ومفاهيم OOP ، فتأكد من مراجعة الفصل 6 قبل التعمق في هذا القسم.

نظراً لأن هذه الواجهة الموجهة للكائنات مدمجة في PHP مع تكوين تثبيت قياسي (يمكنك ببساطة تنشيط امتداد MySQLi في بيئة PHP الخاصة بك) ، كل ما عليك فعله لبدء استخدامه هو إنشاء فئته ، كما في الكود التالي:

```
$db = new mysqli(host, user, password, databaseName);
```

في هذا المثال ، لدينا قاعدة بيانات باسم Library ، وسنستخدم اسم المستخدم الوهمي لـ petermac وكلمة المرور q2w3e9i8u7y1. الكود الفعلي الذي سيتم استخدامه هو:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
```

هذا يتيح لنا الوصول إلى محرك قاعدة البيانات نفسه داخل كود PHP ؛ سنصل على وجه التحديد إلى الجداول والبيانات الأخرى لاحقًا. بمجرد إنشاء هذه الفئة في المتغير $ db ، يمكننا استخدام طرق على هذا الكائن للقيام بعمل قاعدة البيانات الخاصة بنا.

سيبدو مثال موجز لإنشاء بعض التعليمات البرمجية لإدراج كتاب جديد في قاعدة بيانات المكتبة على النحو التالي:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");


$sql = "INSERT INTO books (authorid, title, ISBN, pub_year, available)
 VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1)";
```

```
if ($db->query($sql)) {
 echo "Book data saved successfully.";
} else {
 echo "INSERT attempt failed, please try again later, or call tech support" ;
}

$db->close();
```

أولاً ، نقوم بإنشاء فئة MySQLi في المتغير $ db. بعد ذلك ، نبني سلسلة أوامر SQL الخاصة بنا ونحفظها في متغير يسمى $ sql. ثم نسمي طريقة الاستعلام الخاصة بالفئة وفي نفس الوقت نختبر قيمة الإرجاع الخاصة بها لتحديد ما إذا كانت ناجحة (TRUE) ، ثم نعلق على الشاشة وفقًا لذلك. قد لا ترغب في إعادة الصدى إلى المتصفح في هذه المرحلة ، مرة أخرى هذا مجرد مثال. أخيرًا ، نسمي طريقة close () في الفصل لترتيب الفصل وإتلافه من الذاكرة.

استرجاع البيانات للعرض

في منطقة أخرى من موقع الويب الخاص بك ، قد ترغب في استخلاص قائمة بكتبك وإظهار من هم مؤلفوها. يمكننا تحقيق ذلك من خلال استخدام نفس فئة MySQLi والعمل مع مجموعة النتائج التي تم إنشاؤها من أمر SELECT SQL. هناك العديد من الطرق لعرض المعلومات في المتصفح ، وسنلقي نظرة على مثال واحد لكيفية القيام بذلك. لاحظ أن النتيجة التي تم إرجاعها هي كائن مختلف عن $ db الذي قمنا بإنشائه أولاً. تقوم PHP بإنشاء مثيل لكائن النتيجة وتعبئته بأي بيانات تم إرجاعها.

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
```

```
$sql = "SELECT a.name, b.title FROM books b, authors a WHERE
a.authorid=b.authorid";
$result = $db->query($sql);

while ($row = $result->fetch_assoc()) {
 echo "{$row['name']} is the author of: {$row['title']}<br />";
}

$result->close();
$db->close();
```

هنا ، نستخدم استدعاء طريقة الاستعلام () ونخزن المعلومات التي تم إرجاعها في المتغير المسمى النتيجة $. ثم نستخدم طريقة الكائن الناتج تسمى fetch_assoc () لتوفير صف واحد من البيانات في كل مرة ، ونقوم بتخزين هذا الصف الفردي في المتغير المسمى $ row. يستمر هذا ما دامت هناك صفوف للمعالجة. ضمن حلقة while loop ، نقوم بإلقاء المحتوى في نافذة المتصفح. أخيرًا ، نحن نغلق كلاً من النتيجة وكائنات قاعدة البيانات.

سيبدو الإخراج كما يلي:

**J.R.R. Tolkien is the author of: The Two Towers**

**J.R.R. Tolkien is the author of: The Return of The King**

**J.R.R. Tolkien is the author of: The Hobbit**

**Alex Haley is the author of: Roots**

**Tom Clancy is the author of: Rainbow Six**

**Tom Clancy is the author of: Teeth of the Tiger**

**Tom Clancy is the author of: Executive Orders...**

ملحوظة

تعد multi_query () واحدة من أكثر الطرق المفيدة في MySQLi ، والتي تتيح لك تشغيل أوامر SQL متعددة في نفس العبارة. إذا كنت تريد إجراء INSERT ثم بيان UPDATE استنادًا إلى بيانات مماثلة ، فيمكنك القيام بذلك كله في استدعاء طريقة واحدة - خطوة واحدة.

لقد خدشنا ، بالطبع ، ما تقدمه فئة MySQLi. إذا قمت بمراجعة وثائقها) ، فسترى قائمة واسعة من الطرق التي تشكل جزءًا من هذا الفصل ، بالإضافة إلى كل فئة نتيجة موثقة ضمن نطاق الموضوع المناسب.

سكليتي

SQLite عبارة عن قاعدة بيانات مضغوطة وعالية الأداء (لمجموعات البيانات الصغيرة) و- كما يوحي اسمها- قاعدة بيانات خفيفة الوزن. SQLite جاهز للخروج مباشرة من الصندوق عند تثبيت PHP ، لذا إذا بدا أنه مناسب لاحتياجات قاعدة البيانات الخاصة بك ، فتأكد من قراءتها.

كل تخزين قاعدة البيانات في SQLite يعتمد على الملفات ، وبالتالي يتم إنجازه دون استخدام محرك قاعدة بيانات منفصل. يمكن أن يكون هذا مفيدًا جدًا إذا كنت تحاول إنشاء تطبيق باستخدام قاعدة بيانات صغيرة ولا توجد تبعيات للمنتج بخلاف PHP. كل ما عليك فعله لبدء استخدام SQLite هو الرجوع إليه في التعليمات البرمجية الخاصة بك.

توجد واجهة OOP لـ SQLite ، لذا يمكنك إنشاء مثيل لكائن بالعبارة التالية:

```
$db = new SQLiteDatabase("library.sqlite");
```

الشيء الرائع في هذه العبارة هو أنه إذا لم يتم العثور على الملف في الموقع المحدد ، يقوم SQLite بإنشائه لك. بالاستمرار في مثال قاعدة بيانات المكتبة الخاصة بنا ، فإن الأمر الخاص بإنشاء جدول المؤلفين وإدراج صف عينة داخل SQLite سيبدو مثل المثال 9-2.

مثال 9-2. جدول مؤلفي مكتبة SQLite

```
$sql = "CREATE TABLE 'authors' ('authorid' INTEGER PRIMARY KEY, 'name'
TEXT)";

if (!$database->queryExec($sql, $error)) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Authors was created <br />";
}
```

```
$sql = <<<SQL
INSERT INTO 'authors' ('name') VALUES ('J.R.R. Tolkien');
INSERT INTO 'authors' ('name') VALUES ('Alex Haley');
INSERT INTO 'authors' ('name') VALUES ('Tom Clancy');
INSERT INTO 'authors' ('name') VALUES ('Isaac Asimov');
SQL;

if (!$database->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Authors - OK<br />";
}
```

**Table Authors was createdINSERT to Authors - OK**

**ملحوظة**

في SQLite ، على عكس MySQL ، لا يوجد خيار AUTO_INCREMENT. بدلاً من ذلك ، يجعل SQLite أي عمود يتم تعريفه بـ INTEGER و PRIMARY KEY عمودًا يتزايد تلقائيًا. يمكنك تجاوز هذا السلوك الافتراضي من خلال توفير قيمة للعمود عند تنفيذ جملة INSERT.

لاحظ أن أنواع البيانات مختلفة تمامًا عما رأيناه في MySQL. تذكر أن SQLite هي أداة قاعدة بيانات مختصرة وبالتالي فهي "خفيفة" في أنواع البيانات الخاصة بها ؛ انظر الجدول 9-1 للحصول على قائمة بأنواع البيانات التي يستخدمها.

الجدول 9-1. أنواع البيانات المتوفرة في SQLite

| Data type | Explanation |
| --- | --- |
| Text | Stores data as NULL, TEXT, or BLOB content. If a number is supplied to a text field, it is converted to text before it is stored. |
| Numeric | Can store either integer or real data. If text data is supplied, SQLite attempts to convert the information to numerical format. |
| Integer | Behaves the same as the numeric data type. However, if data of the real type is supplied, it is stored as an integer. This may affect data storage accuracy. |
| Real | Behaves the same as the numeric data type, except that it forces integer values into floating-point representation. |

| Data type | Explanation |
| --- | --- |
| None | This is a catchall data type; it does not prefer one base type to another. Data is stored exactly as supplied. |

قم بتشغيل التعليمات البرمجية التالية في المثال 9-3 لإنشاء جدول الكتب وإدراج بعض البيانات في ملف قاعدة البيانات.

مثال 9-3. جدول كتب مكتبة SQLite

```
$db = new SQLiteDatabase("library.sqlite");


$sql = "CREATE TABLE 'books' ('bookid' INTEGER PRIMARY KEY,
 'authorid' INTEGER,
 'title' TEXT,
 'ISBN' TEXT,
 'pub_year' INTEGER,
 'available' INTEGER,
)";
```

```
if ($db->queryExec($sql, $error) == FALSE) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Books was created<br />";
}

$sql = <<<SQL
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Two Towers', '0-261-10236-2', 1954, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Return of The King', '0-261-10237-0', 1955, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (2, 'Roots', '0-440-17464-3', 1974, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

```
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'Foundation', '0-553-80371-9', 1951, 1);
SQL;

if (!$db->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Books - OK<br />";
}
```

لاحظ أنه يمكننا تنفيذ أوامر SQL متعددة في نفس الوقت. يمكننا أيضًا القيام بذلك باستخدام MySQLi ، ولكن عليك أن تتذكر استخدام طريقة multi_query () ؛ باستخدام SQLite ، يتوفر مع طريقة queryExec (). بعد تحميل قاعدة البيانات ببعض البيانات ، قم بتشغيل الكود في المثال 9-4.

مثال 9-4. حدد سكليتي الكتب

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");

$sql = "SELECT a.name, b.title FROM books b, authors a WHERE a.authorid=b.authorid";
```

```
$result = $db->query($sql);


while ($row = $result->fetch()) {

 echo "{$row['a.name']} is the author of: {$row['b.title']}<br/>";

}
```